AQA qualification training

A-level Computer Science

Focus on Non Exam Assessment

BOOKLET 6

Published date: Summer 2016 version 1.0

Permission to reproduce all copyright materials have been applied for. In some cases, efforts to contact copyright holders have been unsuccessful and AQA will be happy to rectify any omissions of acknowledgements in future documents if required.

A Level Computing Coursework

‘Endless Temple’
A Mobile, Top Down, Adventure Game

AQA Exemplar NEA

Contents

Analysis

Background to the Problem

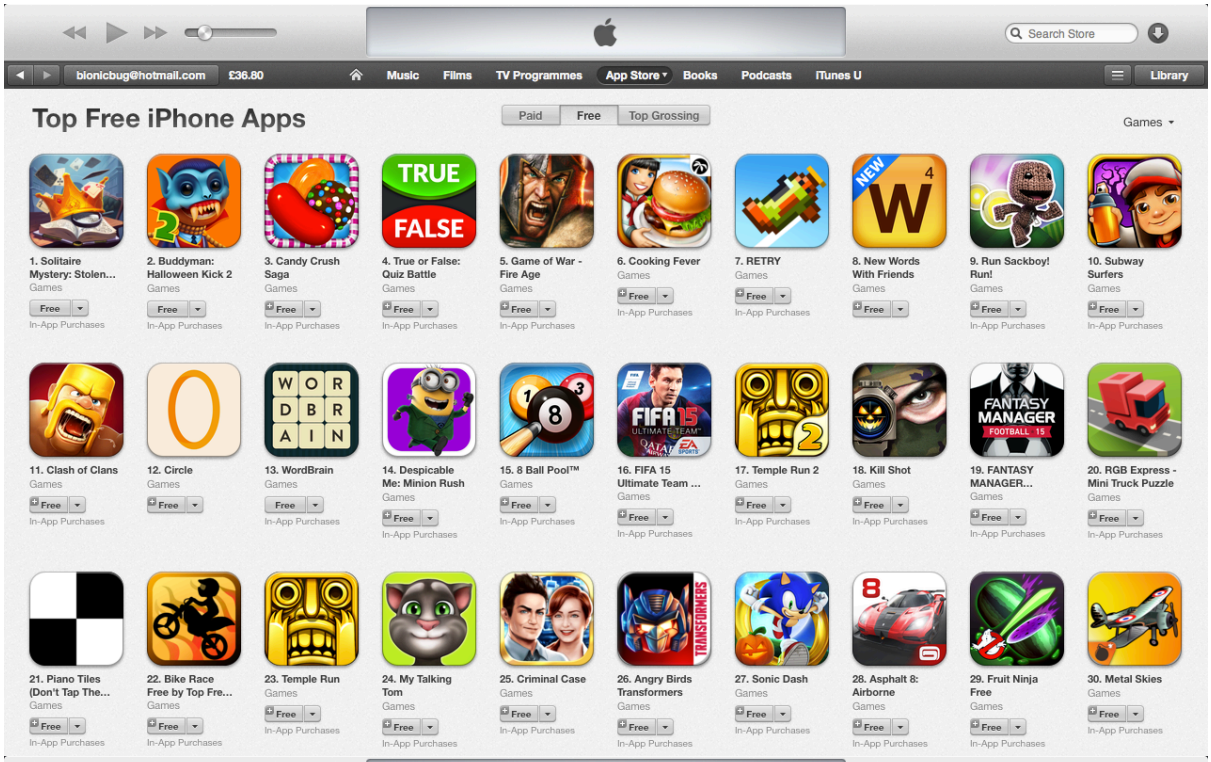
The iTunes app store is a global application distribution service for iOS devices such as the iPhone. Through even a brief look through the iTunes app store charts for games it is evident that fast paced, relatively simple games are the most popular, for example puzzle games like candy crush. This is most likely due to people taking their phone out while waiting for a few minutes to play a quick level or round of a game. Screenshots of the charts are contained on the next page.

I have observed the problem here in that these short paced games are often very simple and start to become boring incredibly quickly. Although through brief investigation this view is unanimously shared, In order to fully understand the problem from other users' points of view I will need to create a couple of surveys. The prospective user is not a company that can simply tell me what to do but rather individual people so I will need to take this into account when consulting the target audience.

A solution to this problem would be a game for smartphones that balances an ability to be played for short periods of time with more variation and interesting gameplay mechanics. This will keep entertaining people for longer as they no longer lose interest in what was previously simple and repetitive gameplay.

Candy Crush Saga[1]

[1] http://3.bp.blogspot.com/-Bnu7JYoCFFA/UP3dbilApRI/AAAAAAAAAmY/wZjlOn1-Xsk/s1600/Screenshot_2013-01-21-12-31-20.png



Screenshots of the iTunes app store charts for games.

The Current System

More complex games exist with similarities to what might be seen on consoles, though obviously these are limited in scope due to the hardware restrictions of a mobile platform. They are impractical for someone who only has a few minutes to play however it may be useful to investigate what features I may be able to include in my app without it becoming similarly impractical.

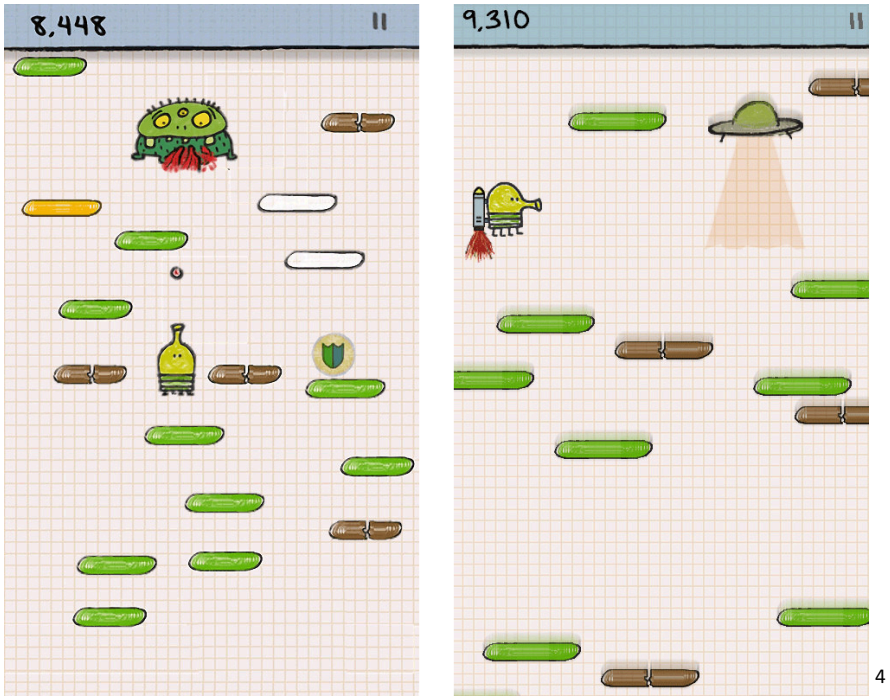
Dungeon Hunter 4, pictured below, is a good example of one of these games with a level of complexity impractical for being able to play for a couple of minutes. The user interface while playing is relatively cluttered and the menu has many tabs including a crafting page and a shop. Crafting takes too much time and along with the shop does not have the same high paced gameplay that is required to entertain the player. By the time the player has finished crafting or shopping they will have run out of time to play. Despite these problems the combat itself is fun and the controls are easy to use, consequently I may gain inspiration from this.

[2]

[2] http://2.bp.blogspot.com/-pW2c6M-tzL4/UXAO8MsYDuI/AAAAAAAABm8/w9scD9Hv7Ak/s1600/Dungeon_Hunter_4_3.jpg

[3]

The simpler games, such as doodle jump pictured below, face the opposite problem. The game is very simple and may entertain the player for a short while, however the repetitiveness and lack of complexity means the player quickly loses interest. On the other hand, the new maps every time and absence of over complexity means it suits the short amount of time available, while the competitive aspect with high scores is an interesting feature I may consider in my solution.



[4]

[3] http://wpuploads.appadvice.com/wp-content/uploads/2013/04/IMG_4002.jpeg

[4] http://www.windowscentral.com/sites/wpcentral.com/files/resource_images/doodle-jump-2-screens.jpg

Prospective Users

The prospective users of this software are busy smartphone users who want to pass the time for short periods that are long enough for the person to become bored, with nothing to do, and yet short enough that a relatively simple pastime is required. I will be targeting the younger end of this set consisting mostly of teenagers so I have collected a group of prospective users from my school to respond to some surveys and discuss what they would want from the application I will be developing. The names of the prospective users I have gathered are .....

User Needs

In order to gain an understanding of whether the target market agree with my initial vision, I proceeded to ask the prospective users the following questions in order to fully realise the demands of a typical user:

1. What sort of things do you look for when browsing mobile game applications?
2. Does the idea of a fast paced, highly replayable game interest you?
3. What interests/doesn't interest you about it?
4. What do you believe is missing from these types of games?
5. Is there anything else you want to add that you feel you haven't talked about in these questions but may be relevant?

From this preliminary survey I have ascertained that I was correct in my identification of the lack of and demand for a fast paced, highly replayable game. I am still left however with the quandary about what the game should be about and how it should be played. It is clear from the results that the game has to have goals and continue to entertain the player with variation in the maps and a distinct difficulty so that it doesn't become boring. A potentially interesting feature was recommended in the survey of being able to level up or buy items, giving the player something to work for.

1) Which of the following genres do you feel you would enjoy most and would fit best in a top-down fast paced mobile game and why?
   a) Fantasy
   b) Modern shooter
   c) Other (please specify)
2) Which of the following do you feel is the best method of making such a game replayable and fast paced?
   a) Set levels
   b) Procedurally generated levels making use of an increasing difficulty factor, such that each level is unique and you won't run out of them
   c) Arcade-style runs where the aim is to get as far as you can before dying then trying to beat highscores on a set map or set of maps
   d) Arcade-style runs where the aim is to get as far as you can before dying then trying to beat high-scores on procedurally generated maps each time such that each time the map is different to before

The results for this survey suggest that a shooter would be best however the responses for the question of how to make the game replayable are mixed. Due to the time constraints on the project I will use a procedure to generate maps so that there is enough variation to create a sense of adventure.

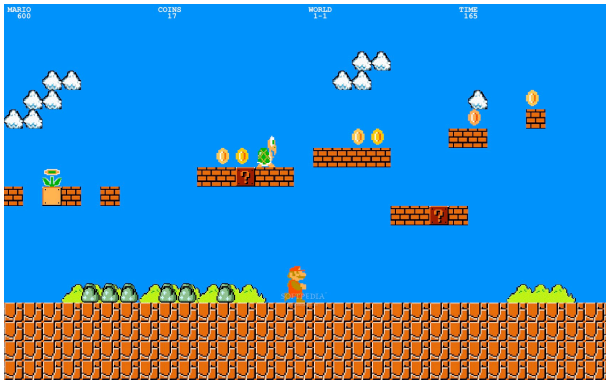
Acceptable Limitations

Bearing in mind the timeframe provided by this project and the limitations of only myself working on the application, a 3D game is impractical. In addition using a 3D environment would indicate and most probably involve a level of gameplay complexity unsuitable for when a user only wants to play for a few minutes at a time, thereby dissuading people from considering downloading it.

In this project I aim to have the game functional however due to the time constraints I will not be able to add as much content, such as textures, animations and features, as I would ideally want. This should not detract much from the gameplay and would not be difficult given lots more time.

In terms of a 2D environment I am presented with two options, a top-down camera style or a side-on camera style. I believe that a side-on camera style severely hinders the room for gameplay elements due to what is usually a fixed route followed by the user. Consequently I have settled on a top-down camera viewpoint.

Top-down camera style[5]



Side-on camera style[6]

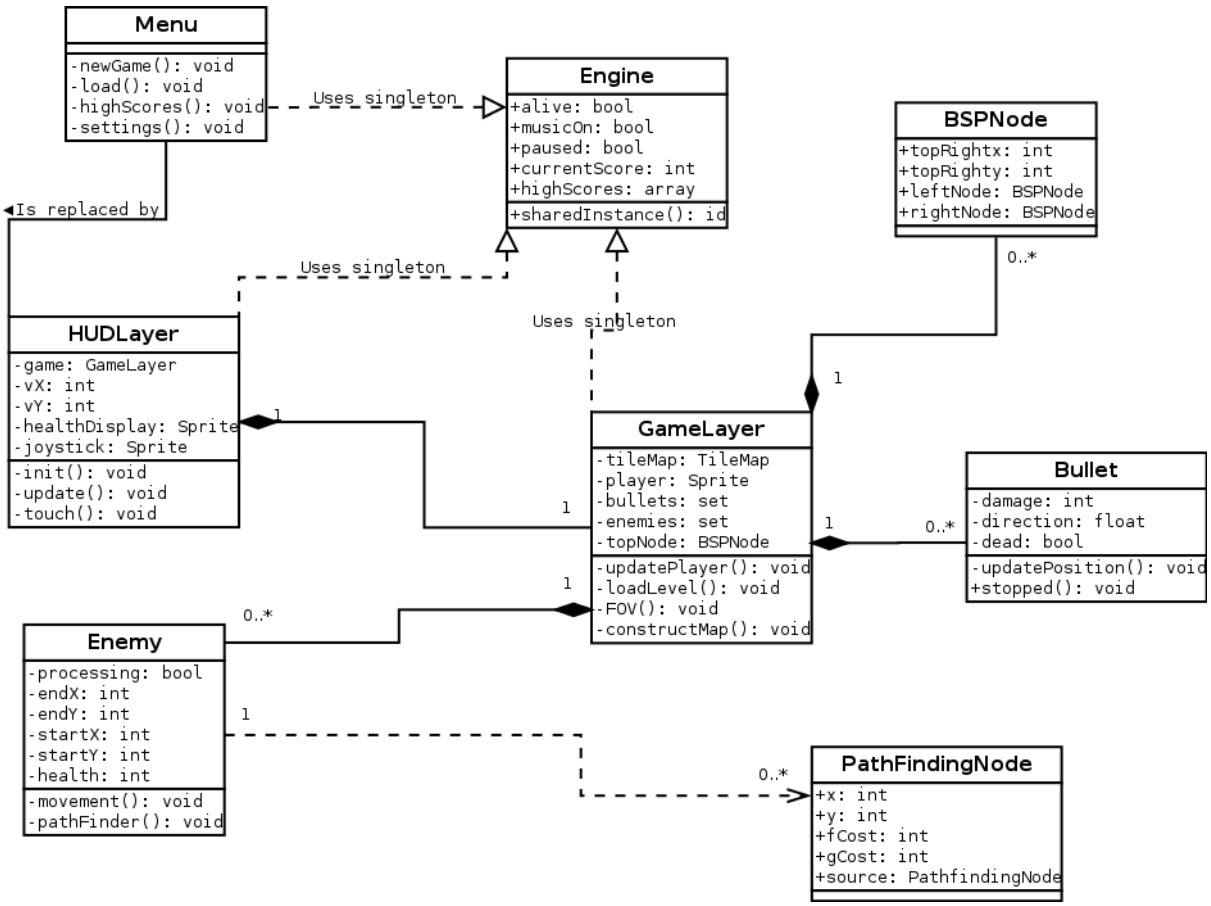
[5] www.cubed3.com/media/2013/April/CTRP_Zelda_scrn01_Ev04.jpg

[6] http://i1-mac.softpedia-static.com/screenshots/Jario_2.jpg

Analysis Key Variables

Name	Purpose	Type	Size	Example
Score	Stores the current score of the player if they quit during a game	Integer	2	5
Health	Stores the health of the player if they quit during a game	Integer	1	87
Ammo	Stores the ammo of the player if they quit during a game	Integer	1	10
Music	Stores whether the user had music on or off	Boolean	1	1
Sfx	Stores whether the user had sfx on or off	Boolean	1	0
Gore	Stores whether the user had gore on or off	Boolean	1	1
PlayerX	Stores the x coordinate of the player if they quit during a game	Integer	2	200
PlayerY	Stores the y coordinate of the player if they quit during a game	Integer	2	252
HighScoreNames	Stores the names of the people on the high score table	Array of strings	Array size 5, string length 10	[‘Dan’,’Dave’,’John’,’Ben’,’Max’]
HighScores	Stores the scores of the people on the high score table	Array of integers	Array size 5, integer length 2	[254,176,132,94,32]
Alive	Stores whether the player quit during a game	Boolean	1	1

Initial Object Analysis Diagram



Objectives

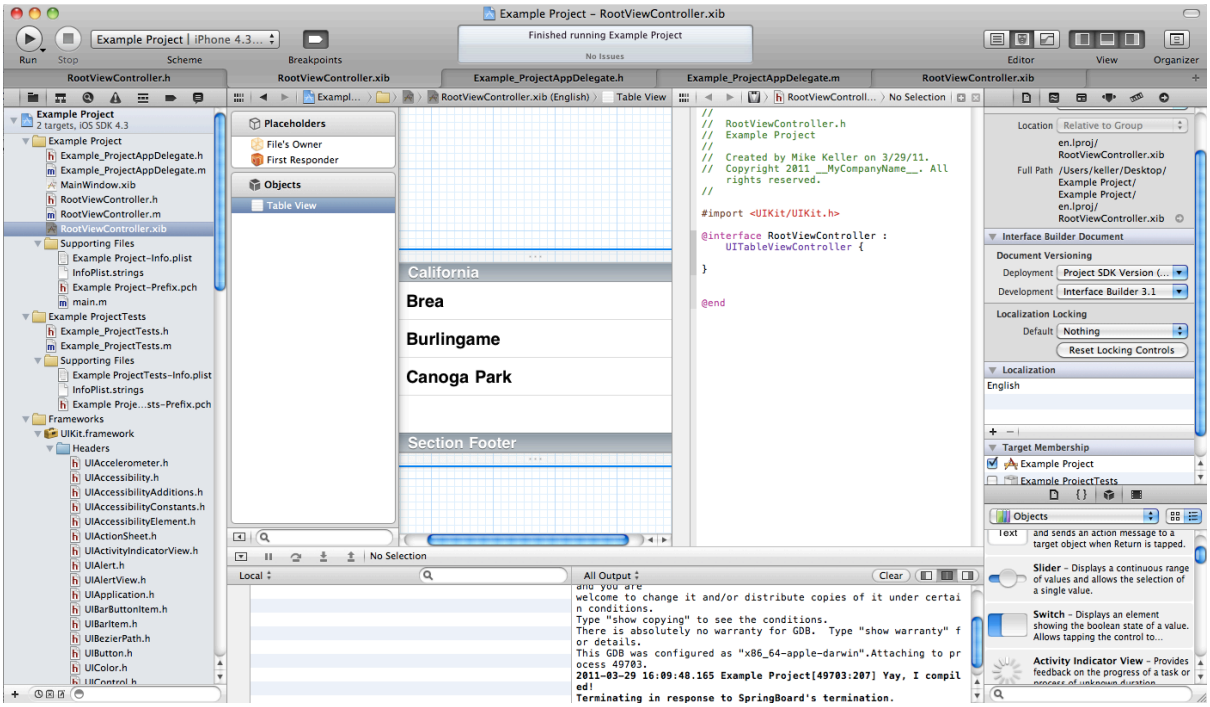
After this research I decided to have a more open discussion with a smaller group of potential end users to propose features they might like to see. The results of this allowed me to have a clear vision of what the game should be like and I have subsequently decided to set down a list of initial objectives for the application in terms of elements that I believe may be of note due to complexity and features that have been highlighted through discussion as being required, they are as follows:

1) A main menu system containing settings
2) An in-game menu allowing people to pause and save their game
3) A system of procedurally generating maps
4) Joysticks that interpret touches on the screen as directions and magnitudes
5) Aiming and movement using the joysticks
6) Enemies with intelligent pathfinding and detection mechanics allowing evasion by the player
7) A line of sight feature shading out areas not explored yet
8) Runtime created bullet objects to be fired by the player and the enemies
9) Collision detection to prevent the player from walking through walls and to stop bullet objects from passing through walls
10) Animations, for example seeing the player and enemies aim, reload and walk
11) There should be no noticeable frame rate drops

In order to solve some of the more complex objectives I have identified a few appropriate algorithms. For the procedural generation of maps the binary space partition algorithm would create maps ideal for the style of game. To enable intelligent pathfinding the enemies could use the A* pathfinding algorithm and for the line of sight feature I have identified a recursive shadow casting technique. As I will be running an update sequence of methods every thirtieth of a second the possibility of having frame rate drops, where for example multiple enemies are executing their pathfinding methods and overloading the processor, means that I may need to use multiple threads and execute these methods in parallel to the main thread.

Potential Solution

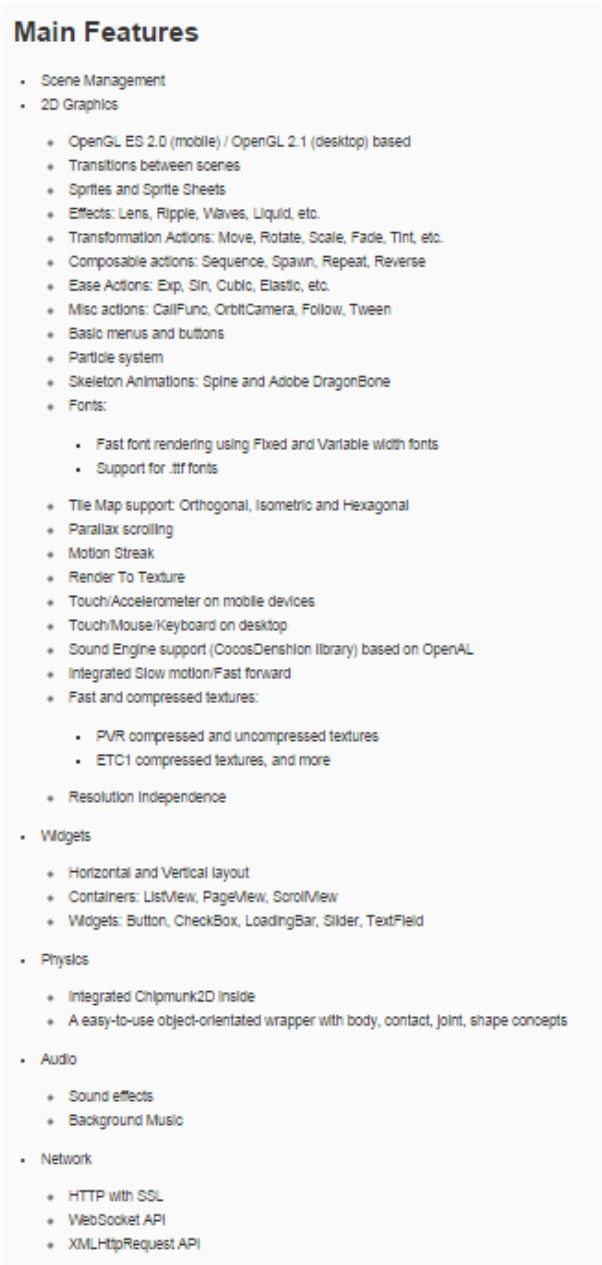
Using XCode to develop the entire application. XCode is designed for developing applications for iPhones, allowing quick testing on devices and easier development for touchscreen applications with copious amounts of documentation and guides. Furthermore an object oriented approach would suit the game in order to easily navigate the program code and the problem of creating a game naturally breaks itself up into objects requiring their own data and methods. Unfortunately XCode has no native support for tilemaps, sprites and animations and programming the support for these would take too long for me to program in addition to programming the application itself.[7]



[7] https://developer.apple.com/xcode/

Using XCode and cocos2d. The cocos2d library adds support for TMX tilemaps, sprites, animations and layers for the game which would mean that I would be able to complete the application in the timeframe of this project. [8]

The following screenshot shows the features of cocos2d-x, cocos2d has most of these features however uses objective c instead of c++.



Main Features

- Scene Management
- 2D Graphics
  - OpenGL ES 2.0 (mobile) / OpenGL 2.1 (desktop) based
  - Transitions between scenes
  - Sprites and Sprite Sheets
  - Effects: Lens, Ripple, Waves, Liquid, etc.
  - Transformation Actions: Move, Rotate, Scale, Fade, Tint, etc.
  - Composable actions: Sequence, Spawn, Repeat, Reverse
  - Ease Actions: Exp, Sin, Cubic, Elastic, etc.
  - Misc actions: CallFunc, OrbitCamera, Follow, Tween
  - Basic menus and buttons
  - Particle system
  - Skeleton Animations: Spine and Adobe DragonBone
  - Fonts:
    - Fast font rendering using Fixed and Variable width fonts
    - Support for .ttf fonts
  - Tile Map support: Orthogonal, Isometric and Hexagonal
  - Parallax scrolling
  - Motion Streak
  - Render To Texture
  - Touch/Accelerometer on mobile devices
  - Touch/Mouse/Keyboard on desktop
  - Sound Engine support (CocosDenshion library) based on OpenAL
  - Integrated Slow motion/Fast forward
  - Fast and compressed textures:
    - PVR compressed and uncompressed textures
    - ETC1 compressed textures, and more
  - Resolution Independence
- Widgets
  - Horizontal and Vertical layout
  - Containers: ListView, PageView, ScrollView
  - Widgets: Button, CheckBox, LoadingBar, Slider, TextField
- Physics
  - Integrated Chipmunk2D inside
  - A easy-to-use object-orientated wrapper with body, contact, joint, shape concepts
- Audio
  - Sound effects
  - Background Music
- Network
  - HTTP with SSL
  - WebSocket API
  - XMLHttpRequest API

[8] http://cocos2d.org/

Evidence of Analysis

Please see the tables of questionnaire results on the following pages. The questions from earlier were the following:

1. What sort of things do you look for when browsing mobile game applications?
2. Does the idea of a fast paced, highly replayable game interest you?
3. What interests/doesn't interest you about it?
4. What do you believe is missing from these types of games?
5. Is there anything else you want to add that you feel you haven't talked about in these questions but may be relevant?

3) Which of the following genres do you feel you would enjoy most and would fit best in a top-down fast paced mobile game and why?
   a) Fantasy
   b) Modern shooter
   c) Other (please specify)
4) Which of the following do you feel is the best method of making such a game replayable and fast paced?
   a) Set levels
   b) Procedurally generated levels making use of an increasing difficulty factor, such that each level is unique and you won't run out of them
   c) Arcade-style runs where the aim is to get as far as you can before dying then trying to beat highscores on a set map or set of maps
   d) Arcade-style runs where the aim is to get as far as you can before dying then trying to beat high-scores on procedurally generated maps each time such that each time the map is different to before

Object Orientation Plan

Due to the fact that I will be developing this program using object oriented programming, a few of the main classes I will need to create are as follows:

HUDLayer

This class will manage the user interface overlay over the game view, including the joysticks, health bar and ammunition display. GameLayer will be initialised from this class and a special update method in GameLayer will be called roughly every thirtieth of a second.

GameLayer

The game layer will control the game view itself, including facilitating movement through the map and displaying the player, enemy sprites and bullet objects. Upon its update method being called GameLayer will be responsible for things like updating the positions of bullets, ordering alerted enemy objects to execute another pathfinding step and updating the player's field of view.

PauseLayer

When the user wants to pause the game, it will transition to the pause layer. On this layer the user will be able to navigate a set of pages involving settings and help in addition to saving their progress through the map so that they can resume playing later.

Bullet

Each time the player or an enemy wants to fire a bullet, a bullet object will be initialised and travel across the screen until it collides with a wall or a character. The properties of each object must therefore include position, rotation and damage dealt on hit.

Enemy

Each enemy will be initialised from the enemy class, this class will contain methods for tasks like pathfinding and checking their field of view. The properties of each enemy will include their health, whether they are alerted to the player's presence and their position.

Documented Design

Introduction

The application I am designing is a two dimensional, top-down shooter. The player has to escape through mazes of randomly generated, inter-connected rooms and gains points for defeating enemies and reaching higher levels. The map will be revealed as the player travels through it and enemies will hunt the player through the map once alerted to their presence.

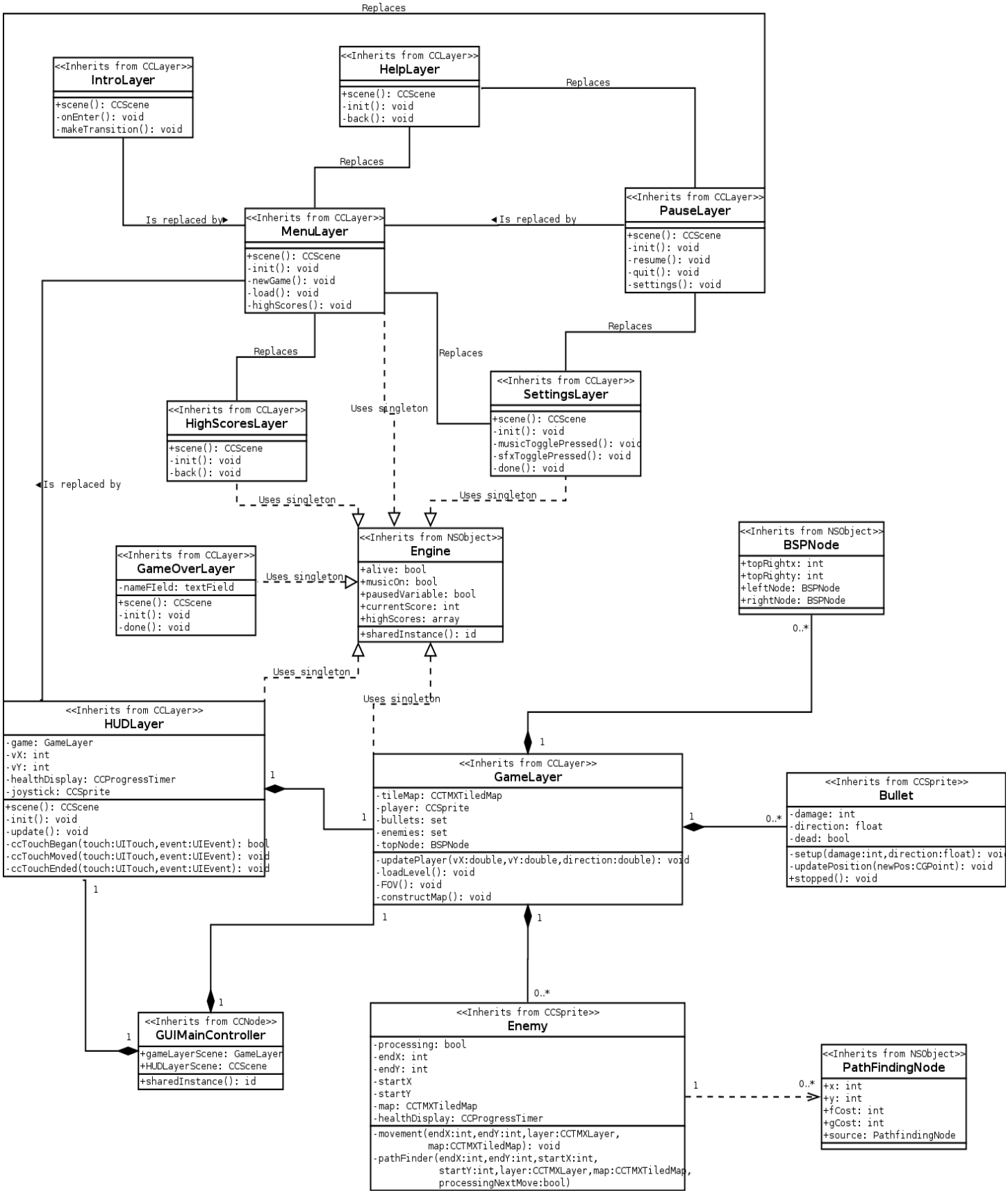
Overall System Design - IPSO Chart

Inputs	Processes	Storage	Outputs
Touches on the touchscreen Saved settings Saved game	Shadow creation Joystick implementation Enemy pathfinding Map generation Menu navigation	Settings property list Saved map array	Sprite positions on the screen User interface elements

The project being designed is a two-dimensional, twin-stick shooter for iPhones and iPods. To achieve this, major items that I will need to design are as follows:

- The menu system
- Map generation
- Enemy pathfinding
- Field of vision shadow creation
- The user interface

Object Analysis Diagram



Modular System Structure

The system will be broken up into the menu pages and the game view itself. The menus will be navigable through the use of buttons and each page will be its own distinct class. From the main menu and the pause screen the user can navigate to the game view, where the screen shows the local area of the map. The joysticks and other user interface elements will be overlaid over the map. Both the overlay and the map will be distinct classes too. Each enemy and bullet fired will be instances of classes and will be managed by the class controlling the map.

Record Structures

The data I will be storing are a list of variables and the map such that the user's settings are stored when they close the application and the current state of the game is stored so that it can be loaded again if the user quits the game and wants to continue later.

The variables will be stored in a property list file, consisting of keys and values whereas the map will be stored by serialising an array of the tile IDs. The enemy data will also be serialised so that they reappear in the correct place with the correct health. These files will be stored in the application's data container in its sandbox memory.

Data Requirements

Name	Purpose	Type	Size	Example
Score	Stores the current score of the player if they quit during a game	Integer	2	5
Health	Stores the health of the player if they quit during a game	Integer	1	87
Ammo	Stores the ammo of the player if they quit during a game	Integer	1	10
Music	Stores whether the user had music on or off	Boolean	1	1
Sfx	Stores whether the user had sfx on or off	Boolean	1	0
Gore	Stores whether the user had gore on or off	Boolean	1	1
PlayerX	Stores the x coordinate of the player if they quit during a game	Integer	2	200
PlayerY	Stores the y coordinate of the player if they quit during a game	Integer	2	252
HighScoreNames	Stores the names of the people on the high score table	Array of strings	Array size 5, string length 10	[‘Dan’,’Dave’,’John’,’Ben’,’Max’]
HighScores	Stores the scores of the people on the high score table	Array of integers	Array size 5, integer length 2	[254,176,132,94,32]
Alive	Stores whether the player quit during a game	Boolean	1	1

Validation Required

Due to my application being a mobile application, I have a large amount of control over what inputs the user can give. Whereas a computer may have a keyboard permanently connected with all of the potential key inputs on it, I can choose when I want a keyboard to appear and what specific keys that keyboard will have. Furthermore when there is not a keyboard the elements that the user can interact with, such as the joysticks and the menu items, all have been specifically designed by myself to do one task, meaning I do not need to validate what the user has pressed because if the element is there the user should be able to use it.

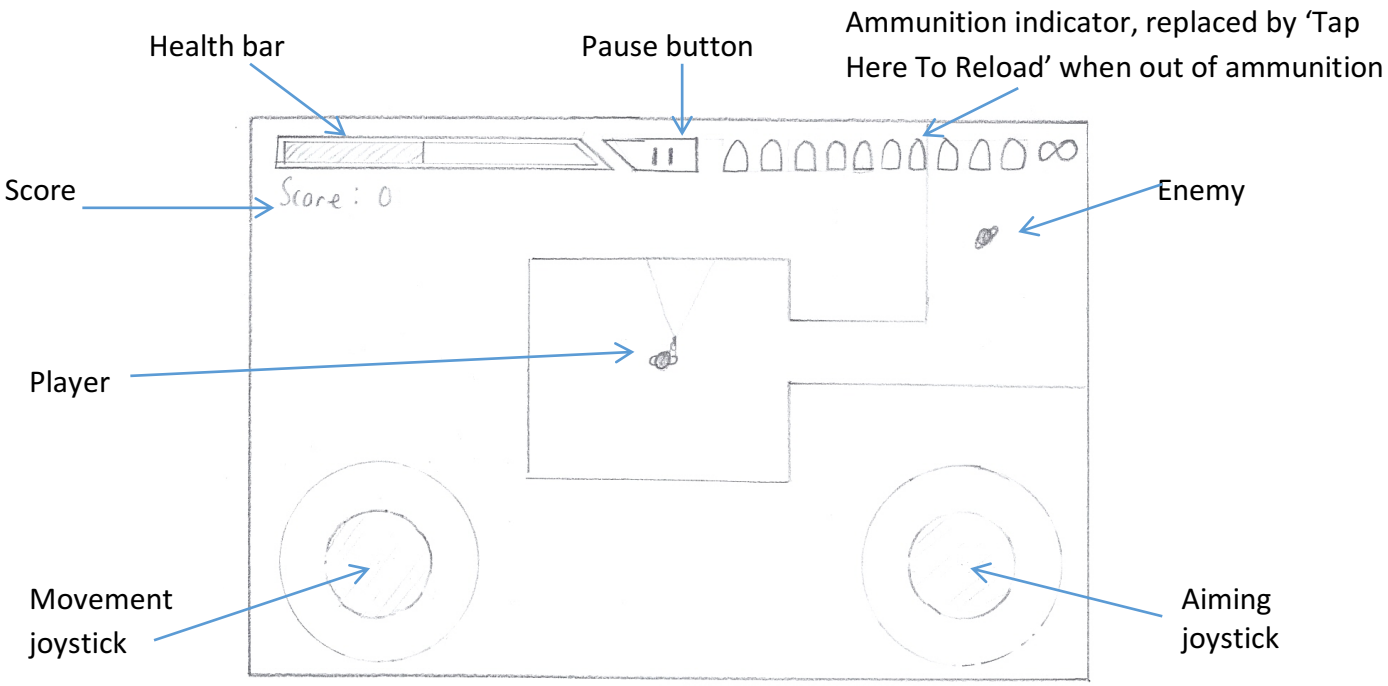
The only time I present a keyboard to the user is when they are entering their name. The validation for this is that the name field cannot be empty or contain a name too large otherwise the name will cause the high-score screen to crash. If the name is too long or not present and the user taps 'Done' an error message will show informing the user of their mistake.

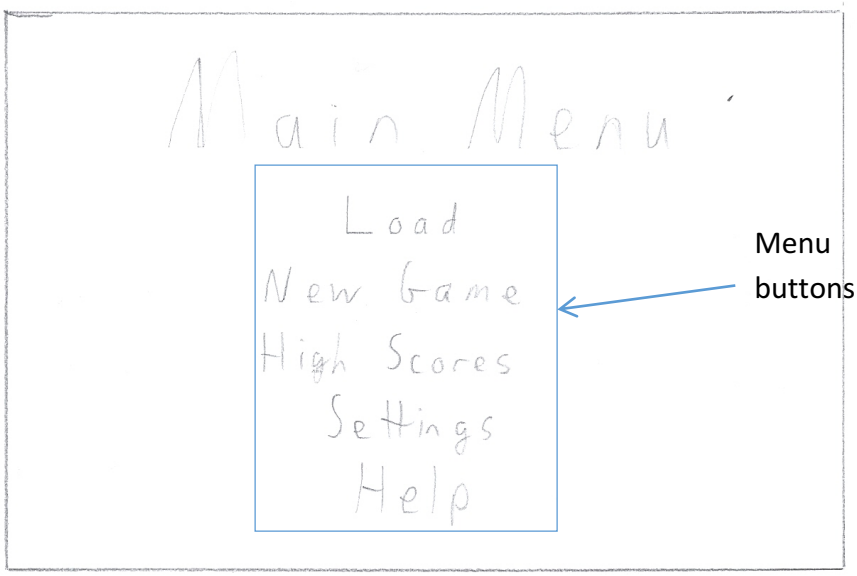
Human-Computer-Interface Rationale

The users of the application that I have identified have small amounts of time to play the game. They will download the application and want to be able to play instantly so the user interface should be very intuitive and uncomplicated.

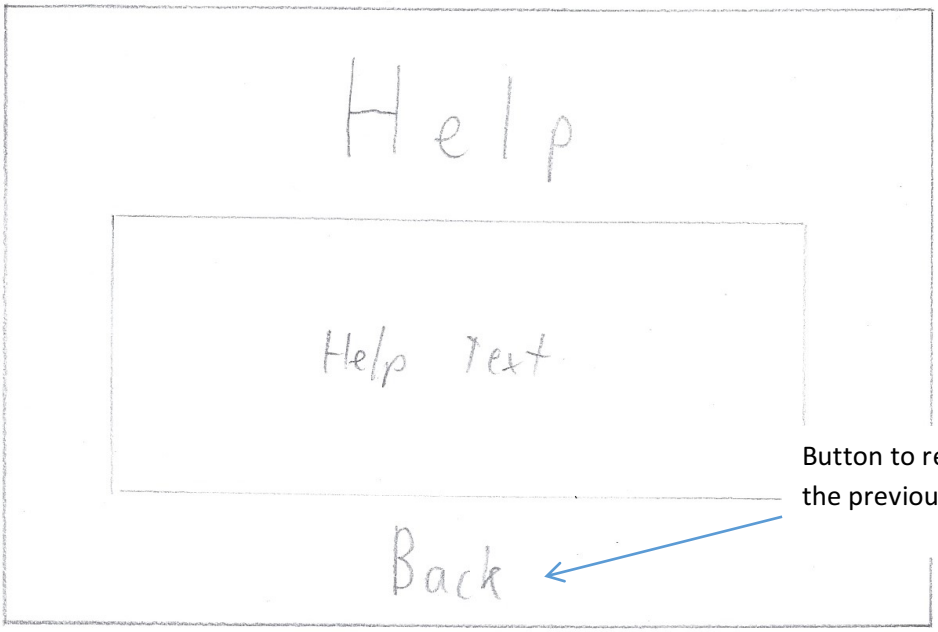
To satisfy this group of users, the menus will have clear buttons, clearly marked exit buttons and a help section as a last resort if people are unable to work out how to play. As I am using discrete user interface elements like joysticks and buttons there should be little to no risk of errors occurring in using the user interface.

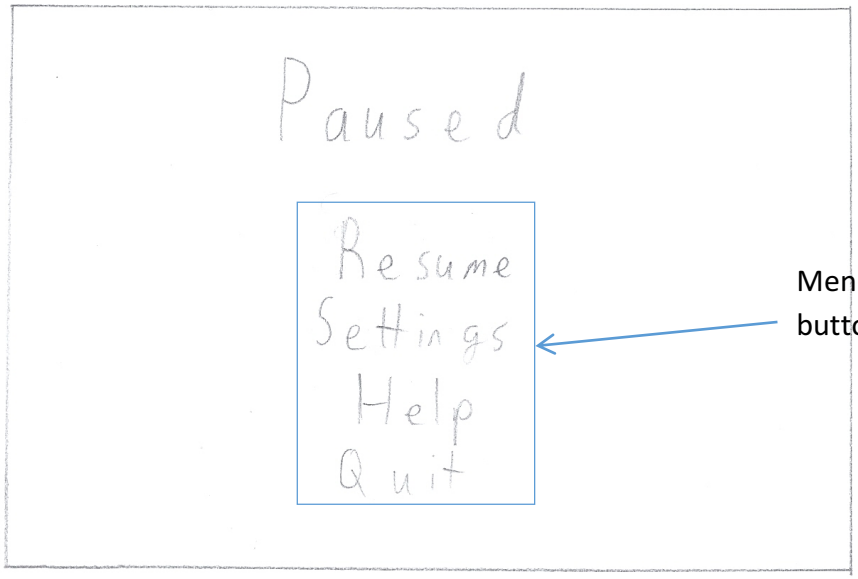
User Interface (UI) Design

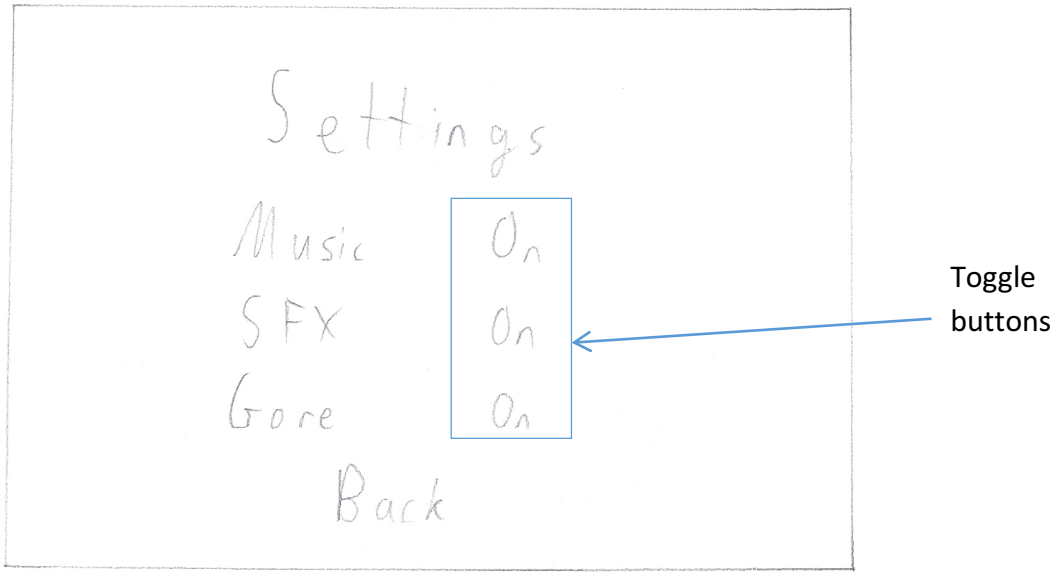




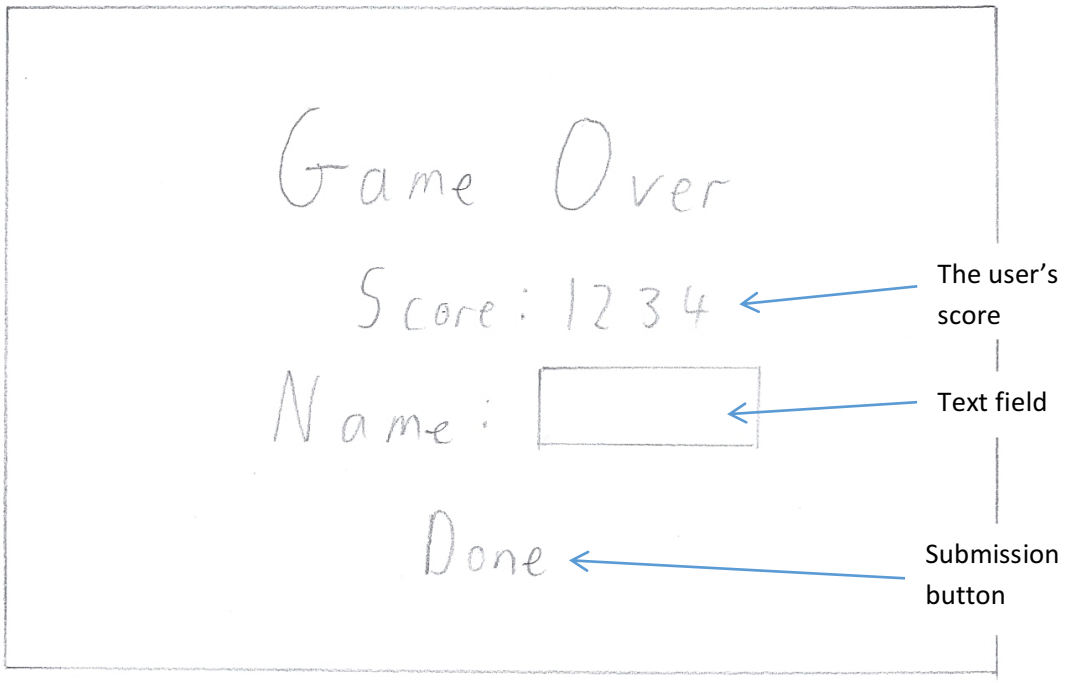

High Scores
1 Dan 1 2 3 4 5
2 Geoff 1 1 2 3 4
3 Dave 1 0 1 2 3
4 Tom 9 4 2 1
5 Peter 5 6 2 4
Back
Button to return to the previous menu


Help
Help Text
Back
Button to return to the previous menu





Toggling the buttons on the settings menu will change Boolean variables, stored in a singleton. This means they can be accessed from the other classes, saved and loaded.


Game Over
Score: 1234
Name:
Done
The user's score
Text field
Submission button

Description Of The Game Process

The game can be split into three sections: Initialisation, the main game phase and the end. Each section will have processes associated with it, run in a set order or reacting to the user's input.

Initialisation

This section consists of a set of processes executed in a specific order to prepare the game.

1. Create the rooms using the binary space partition algorithm.
2. Set the player's initial position in the centre of one of the rooms.
3. Place enemies by dividing each room into set sized squares and for each square placing an enemy in that square if a random number is high enough and if it is far enough from the player's spawn point.
4. Link the rooms with corridors then link those rooms together with corridors recursively.
5. Change the tiles immediately outside the rooms to walls.
6. Set the exit in the middle of a random room, but not in the player's spawn room.
7. Player is given a full magazine of ammunition, the user can not run out of ammunition but they have to reload after using up their magazine of 10 bullets.
8. Player is given 100% health.

Main Game Phase

In this section there will be a combination of both a set of processes that are executed every 30^{th} of a second and procedures that are called in response to touches by the user.

1. Check whether any of the enemies have attacked the player and if so decrease the size of the player's health bar. If the player has died move to the end of the game section.
2. Move the bullets according to the direction they were fired from. Stop any bullets that hit walls or enemies and decrease the hit enemies' health bars. If the enemies have no health left increment the user's score.
3. Decrease the angle between the aiming indicators if the player is aiming to show the angle through which the bullets may randomly fire and adjust their length.
4. Move the player character in the map according to the input from the movement joystick, a small nudge on the joystick should translate to a small movement of the character. If the player is going to move into a wall, move the player as close to the wall as possible then only apply the component of the velocity perpendicular to the wall. When the player moves, move the region visible by the user such that the player character remains at the centre of the screen.
5. If the player has moved onto a new tile execute the shadow-casting algorithm to reveal the tiles not blocked by walls.

6. If the player has moved onto the stairs to exit the level, clear the map then create a new map as in the initialisation section, give the player 100% health and increment the user's score.
7. Check whether each enemy can see the player or have previously seen the enemy then either move it directly towards the player if there is an uninterrupted path between them otherwise execute a path finding algorithm to navigate it around walls.

- If the user taps the pause icon, stop running the main game phase procedures every 30th of a second and open the pause menu.
- If the user taps the ammunition at the top right of the screen or the reload indictor when it appears run the reload animation then update the ammunition indicator to show that the player has 10 bullets. The player cannot aim or fire during this animation.
- If the user touches the right joystick and drags it, resolve the touch location Cartesian coordinates into a direction and rotate the player character to this direction. Present the aiming indicators and run the player aim animation if the player is not reloading.
- If the user releases the right joystick create a bullet object at the player's position that will move in the direction the player is aiming and run the firing animation.
- If the user touches the left joystick and drags it, adjust the touch location Cartesian coordinates into horizontal and vertical components from the centre of the joystick. These components will be proportional to the distance moved by the player each 30th of a second.

The End Of The Game

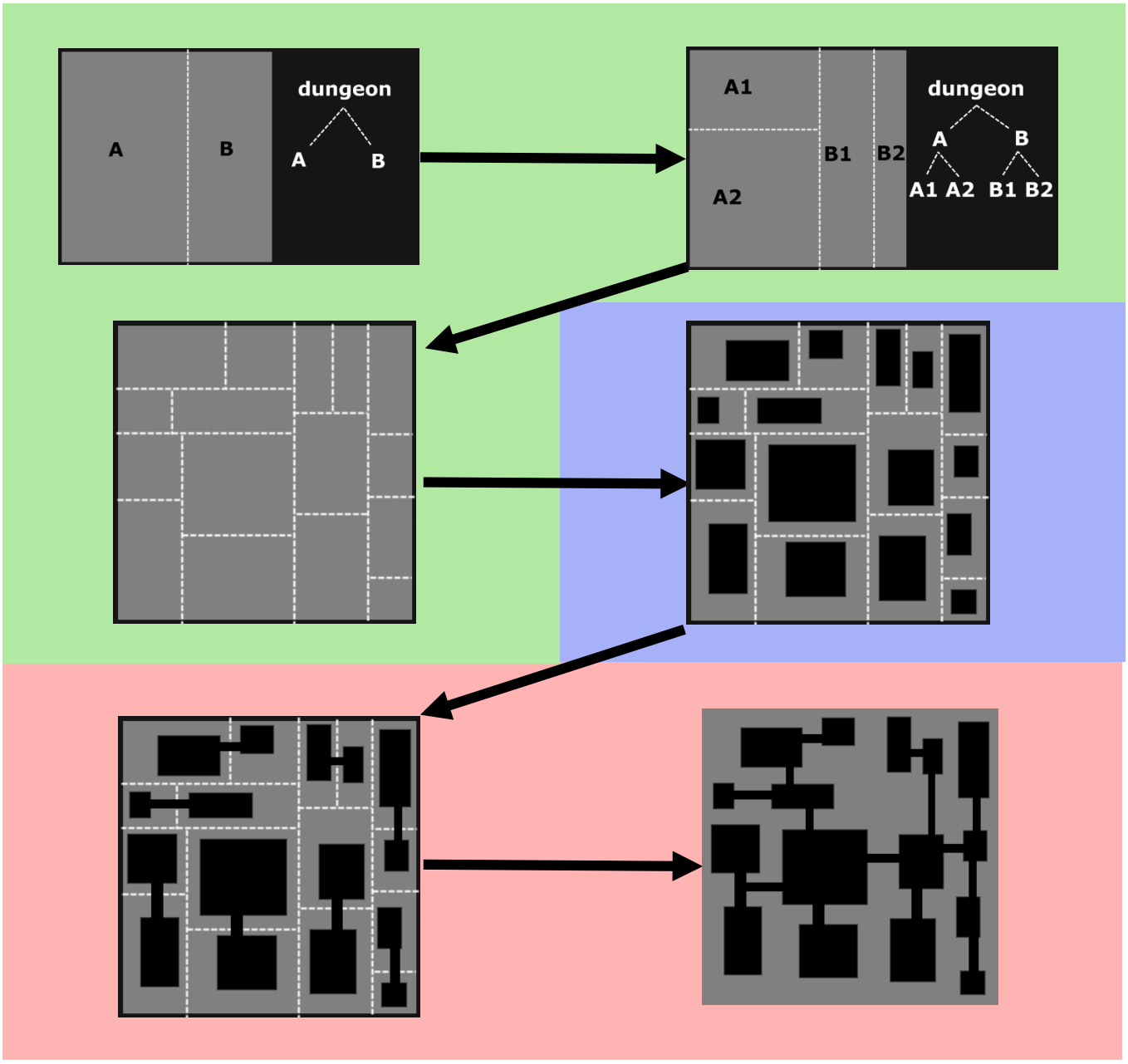
When the player's health is completely depleted the game is over and the following processes are executed.

1. Stop executing the main game phase processes every 30th of a second.
2. Clear the map to prepare for a new map to be generated.
3. Display the game over screen, containing the text box for the user to enter their name.
4. When the user taps done update the high-scores if the user's score was high enough, placing the name that the user entered and their score in the correct position in the leader-board.
5. Return to the main menu.

Algorithms

Generating Maps

This algorithm generates maps consisting of rooms of varying sizes and seemingly random positions such that each map played by the user will be different. Mostly it relies on binary space partitioning. The diagrams shown below illustrate the steps in the algorithm and for each step I have included some pseudo code illustrating the main parts of the step.



1

[1] http://www.roguebasin.com/index.php?title=Basic_BSP_Dungeon_generation

Step 1 (green) – Splitting

On each 'split' call the Node inputted as the parameter is split either horizontally or vertically in a random position 20% to 80% along so that nodes are not created that are so long that the player becomes bored. Once the maximum tree depth has been reached or the node being considered is too small to split again, the node does not call the split function so the sub-tree ends at that node. This recursive function is pre-order as the region is progressively divided from the root node down.

```
Left child node bottom left coordinate = Node bottom left coordinate
Right child node top right coordinate = Node top right coordinate
Select random integer between 0 and 60
Convert integer to a fraction between 0.2 and 0.8

if (random Boolean value is true) //vertical split
  newX = a position along the width of the room defined by the random fraction
  Left child node top right x coordinate = newX
  Right child node bottom left x coordinate = newX
  Left child node top right y coordinate = node top right y coordinate
  Right child node bottom left y coordinate = node bottom left y coordinate
else
  newY = a position along the height of the room defined by the random fraction
  Left child node top right y coordinate = newY
  Right child node bottom left y coordinate = newY
  Left child node top right x coordinate = node top right x coordinate
  Right child node bottom left x coordinate = node bottom left x coordinate

Split the left child node
Split the right child node
```

Step 2 (blue) – Creating rooms

A room is created within the boundaries of each node. The room's boundaries are randomly set in the region between the boundaries of the node and 25% in, enabling variation in size while still creating large enough rooms for the player.

Select random integer between 0 and 25
Convert random number to fraction between 0.75 and 1
newVal = a position along the width of the node defined by the random fraction
Node top right x coordinate = newVal;

Select random integer between 0 and 25
Convert random number to fraction between 0.75 and 1
newVal = a position along the height of the node defined by the random fraction
Node top right y coordinate = newVal;

//the process for setting the opposite vertex's coordinates is similar

Step 3 (red) – Linking rooms

This step links the rooms generated by the previous algorithm together with paths such that the player can move from room to room. Depending on whether the initial split that created the nodes in question was horizontal or vertical, either the y or x coordinate is set such that the function begins creating the path at a point between the two nodes. The other coordinate is determined randomly within the maximum extremities shared by both subtrees being linked. The recursive function is post order so the leaves are linked first, then the linked leaves are linked together as the algorithm ascends the tree.

The following pseudocode covers the code for linking nodes created by a vertical split, as the code for linking nodes created by a horizontal split is incredibly similar.

```
if (left child node exists)
    link left child node
if (right child node exists)
   link right child node
if (node was split vertically)
   find the top y coordinate of the common region between the two areas to be linked
   find the bottom y coordinate of the common region between the two areas to be linked
   select random integer r between 0 and 90
   convert random number r to within the region 0.05 to 0.95
   select a y coordinate at the fraction r of the height of the common region

   x = the coordinate at which the node was split
   loop
      if (tile at (x,y) is walkable)
         break out of the loop
      set the tiles at (x,y), (x,y-1) and (x,y+1) to walkable tiles
      increment x

   x = the coordinate at which the node was split
   loop
      if (tile at (x,y) is walkable)
         break out of the loop
      set the tiles at (x,y), (x,y-1) and (x,y+1) to walkable tiles
      increment x
```

Recursive Shadowcasting

This algorithm updates the shadow layer, removing shadow tiles covering the tiles that can be seen by the player. In order to achieve this, the algorithm splits the area around the player into 8 sections and calculates the shadows for each one individually. To reduce the size and repetition of the program code, the algorithm is given four numbers (xx,yy,xy and yx). These numbers allow the coordinates of the first octant to be translated into the coordinates of each of the 8 octants and processed in exactly the same way.

To determine which cells should be shaded in the octant being processed, the algorithm moves along a row at a time making sure each tile is walkable. When the algorithm reaches a tile that is not walkable it calls itself using the gradient to the bottom right corner of the collidable tile as its top limit then the current call carries on to the top extreme of the collidable tiles and adjusts the bottom limit of the current scan to the gradient from the origin of the shadowcast to the top left corner of the last tile.

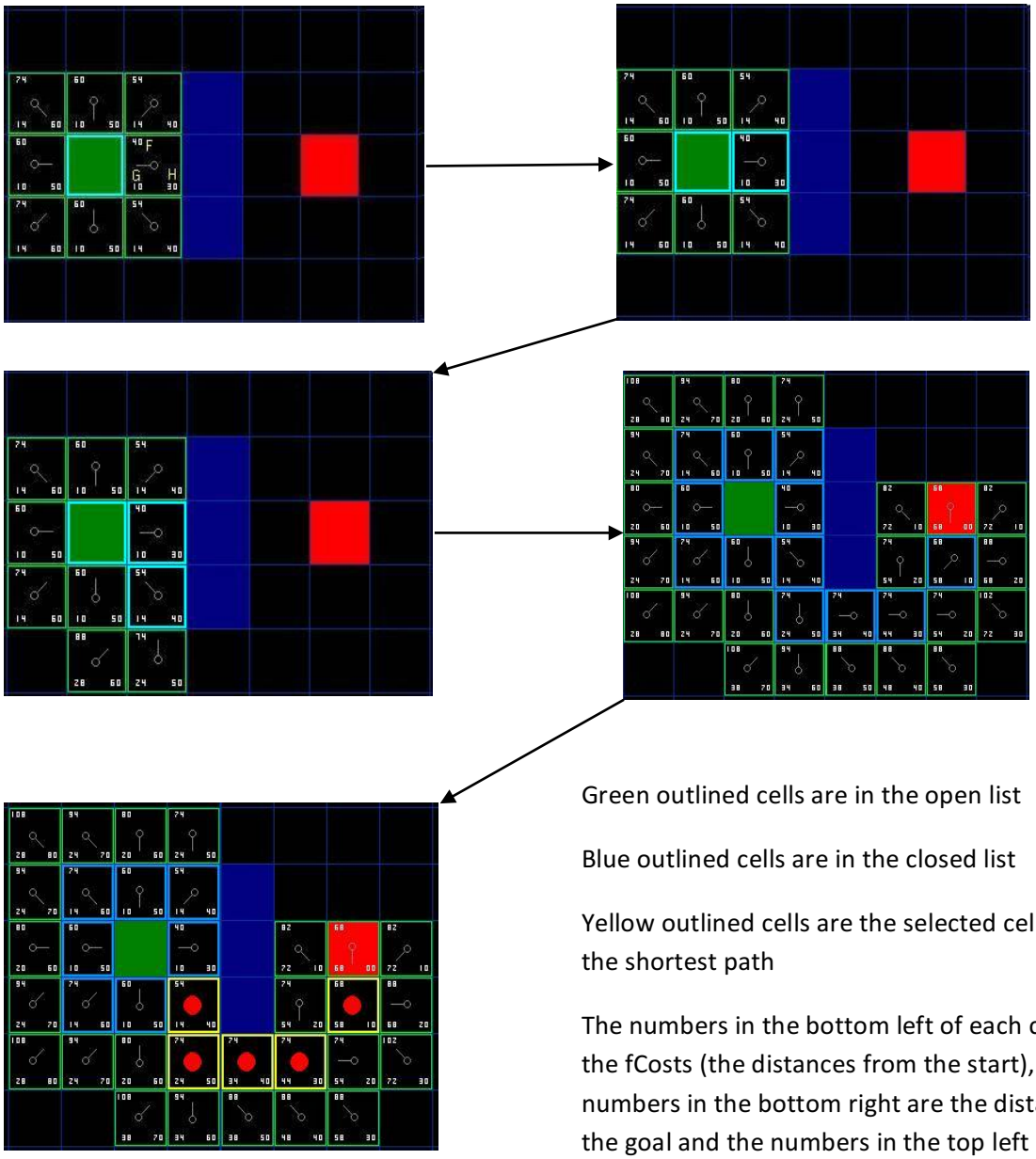
Pseudo code

```
//Cast light in each octant
castLight(1,0,1,0,0,1) //ENE octant
castLight(1,0,1,0,1,1,0) //NNE octant
castLight(1,0,1,0,-1,1,0) //NNW octant
castLight(1,0,1,-1,0,0,1) //WNW octant
castLight(1,0,1,-1,0,0,-1) //WSW octant
castLight(1,0,1,0,-1,-1,0) //SSW octant
castLight(1,0,1,0,1,-1,0) //SSE octant
castLight(1,0,1,1,0,0,-1) //ESE octant
```

```
void castLight(row, startGradient, endGradient, xx, xy, yx, yy)
  previousCollidable = false
  newStartGradient = 0
  for (x = row; x <= 20 and previousCollidable = false; x++)
    for (y = 0; y <= x; y++)
      topGradient = gradient to top left corner of the tile
      bottomGradient = gradient to bottom right corner of the tile
      if (not in a shadow)
        if (reached the bottom of a shadow or reached the line y=x)
          break out of the for loop
        // coordinate transformation
        tempX = player.x + (x * xx) + (y * xy)
        tempY = player.y + (x * yx) + (y * yy)
        if (tile (tempX,tempY) is within a radius of 20 tiles)
          set tile as visible
        if (previousCollidable)
          if (tile collidable at (tempX,tempY))
            newStartGradient = topGradient
          else
            previousCollidable = false
            startGradient = newStartGradient
        else
          if (tile collidable at (tempX,tempY) and x < 20)
            previousCollidable = true
            if (startGradient <= bottomGradient)
              castLight(x + 1, startGradient, bottomGradient, xx, xy, yx, yy)
            newStartGradient = topGradient
```

Pathfinding

This algorithm is based on the A* pathfinding algorithm and is used such that the enemies know where to move in order to get closer to the player in a map that has many walls blocking them. As this algorithm inevitably requires a large processing time it should use multiple threads in order to work out which tile for the enemy to move to once the enemy has reached the tile it is heading for currently, while it is moving towards that tile.



Green outlined cells are in the open list

Blue outlined cells are in the closed list

Yellow outlined cells are the selected cells for the shortest path

The numbers in the bottom left of each cell are the fCosts (the distances from the start), the numbers in the bottom right are the distances to the goal and the numbers in the top left are the gCosts (the sum of the previous two numbers)

The cells with the lowest gCosts are selected each time.

[2] http://www.policyalmanac.org/games/aStarTutorial.htm

Pseudo code

```
Processing = true
if (there is an uninterrupted path between the enemy and the player from 4 points on each extreme of the enemy)
        Simply move towards the player
else
  Create pathfinding node startnode at enemy position
  Add startnode to openlist
  While there are nodes in the openlist
  currentNode = node in the open list with the lowest gcost
  If currentNode is at the player's position
        Travel backwards through the currentNode's source node and its source node until you have reached the node adjacent to the startnode.
        If processing next move
                Set next destination
                NextProcessed = true
                nextSimple = false
        else
                set destination
  else
        add currentNode to closedlist
        remove currentNode from openList
        for each surrounding node around currentNode
                if node not in openList or closedList and not blocked
                        node's source = currentNode
                        node's fCost = currentNode's fCost + distance between node and currentNode
                        node's gCost = fCost + distance to player position
  processing = false
```

Class Definitions

Below are shown the class definitions of the main classes in the program, containing the main attributes and methods. An asterisk (*) for an initial value indicates a pointer to an object.

HUDLayer

This class inherits from CCLayer

Access Type	Field Name	Field Type	Initial Value	Description
Private	game	GameLayer*	*	Pointer to the gameLayer object
Private	vX	NSNumber*	0	Player's horizontal component of their velocity
Private	vY	NSNumber*	0	Player's vertical component of their velocity
Private	direction	NSNumber*	0	Direction the player is facing
Private	aiming	NSNumber*	0	Boolean for whether the user is holding down the aim button
Private	amountAimed	NSNumber*	0	The amount of time (in ticks) the user has held the aim button
Private	health	NSNumber*	100	The health of the player

Access Type	Method Name	Parameters	Return Values	Description
Private	init		A HUDLayer object	Initialises a HUDLayer object and returns it
Private	ccTouchBegan	touch, event		Called when the touchscreen initially detects a touch
Private	ccTouchMoved	touch, event		Called when the touchscreen detects that the touch has moved
Private	ccTouchEnded	touch, event		Called when the user removes their finger from the screen
Private	update			Called every tick and calls the methods which need to be called frequently

GameLayer

This class inherits from CCLayer

Access Type	Field Name	Field Type	Initial Value	Description
Private	tileMap	CCTMXTiledMap*	*	Pointer to the main tilemap
Private	shadowMap	CCTMXTiledMap*	*	Pointer to the shadow tilemap covering the main tilemap
Private	background	CCTMXLayer*	*	Pointer to the layer of the main tilemap containing the ground and wall tiles
Private	enemies	NSMutableSet	null	Set containing pointers to the enemy objects so they can be updated
Private	bullets	NSMutableSet	null	Set containing pointers to the bullet objects so they can be updated
Private	trail1	CCProgressTimer*	*	Pointer to one of the lines that indicate where the player is aiming
Private	trail2	CCProgressTimer*	*	Pointer to the other line that indicates where the player is aiming

Access Type	Method Name	Parameters	Return Values	Description
Private	init		A GameLayer object	Initialises a GameLayer object and returns it
Private	setViewPointCenter	position		Moves the centre of the camera to the player's position
Private	checkForBulletCollision			Iterates through the bullet set checking for collisions with walls and enemies
Private	updatePlayer	vX, vY, direction		Updates the player's position on the map and rotation on the screen
Private	loadLevel	entry, map		Loads a level or calls the map generation method
Private	showTrail	direction, trailNumber		Updates the position and rotations of the lines showing where the player is aiming
Private	FOV			Updates the shadow tilemap using the shadowcasting algorithm, revealing the tiles the player can see

Bullet

This class inherits from CCSprite

Access Type	Field Name	Field Type	Initial Value	Description
Private	damage	NSNumber*	0	The amount of damage to be done by the bullet on hit such that it can be modified from the GameLayer class.
Private	direction	NSNumber*	0	The direction the bullet is heading
Private	dead	NSNumber*	0	A Boolean for whether the bullet is still moving or has hit something
Private	vX	NSNumber*	0	The horizontal component of the bullet's velocity
Private	vY	NSNumber*	0	The vertical component of the bullet's velocity

Access Type	Method Name	Parameters	Return Values	Description
Private	init		A bullet object	Initialises a Bullet object and returns it
Private	setup	Damage, direction		Sets the damage, rotation and horizontal and vertical components of velocity of the Bullet
Private	updatePosition	newPositon		Moves the bullet sprite to the position specified by the parameter
Private	stopped			Sets the horizontal and vertical components of velocity to zero and sets the 'dead' Boolean value to true
Private	futurePosition		The coordinates of the bullet after the next movement	Returns the coordinates of the bullet after the next movement such that collisions can be checked for before the bullet sprite appears inside the wall or enemy

Enemy

This class inherits from CCSprite

Access Type	Field Name	Field Type	Initial Value	Description
Private	dead	NSNumber*	0	A Boolean for whether the enemy has been killed or not
Private	health	NSNumber*	0	The health of the enemy
Private	activated	NSNumber*	0	A Boolean for whether the enemy has detected the player
Private	processing	NSNumber*	0	A Boolean for whether the next move is being determined
Private	nextSimple	NSNumber*	0	A Boolean for whether there is an uninterrupted line to the player, meaning pathfinding is unnecessary
Private	nextProcessed	NSNumber*	0	A Boolean for whether the move after the move being made has been determined
Private	healthDisplay	CCProgressTimer*	*	Pointer to the healthbar sprite above the enemy

Access Type	Method Name	Parameters	Return Values	Description
Private	initWithHealth : Damage:	Health, damage		Sets many of the fields to their initial values on the enemy's spawn and initialises the health bar
Private	hit	Damage, direction, map, background layer		Activates the enemy if they haven't already seen the player, reduces the enemy's health, knocks the enemy back
Private	movement	Destination x and y values, map, background layer		If the enemy is not activated, checks whether the enemy can see the player and activates the enemy if necessary. If the enemy is activated calls pathFinder on a different thread. Also calls pathFinder to calculate the next move after the move currently being made.
Private	pathFinder	endX, endY, startX, startY, background layer, map, processingNextMove		Checks whether the enemy can simply walk to the player otherwise executes an implementation of the A* pathfinding algorithm using a heuristic estimate
Private	directLine	playerPos, background layer, map	Boolean for whether there is an unblocked line to the player	Checks whether the enemy can see the player, meaning the player is in the enemy's sight cone and is not blocked by any walls.
Private	hunt			After pathFinder has found the next point to move to, hunt is called to move the enemy gradually over there to avoid calling pathFinder again unnecessarily.
Private	spaceIsBlocked	x, y, background layer, map	Boolean for whether the enemy could move to the point specified	Checks whether any of the 4 tiles surrounding the point specified are walls. If none of them are returns NO otherwise returns YES.
Private	nodeInArray	Array, x, y	PathfindingNode	Checks whether a pathfinding node exists in the specified array at the coordinates specified.

Testing

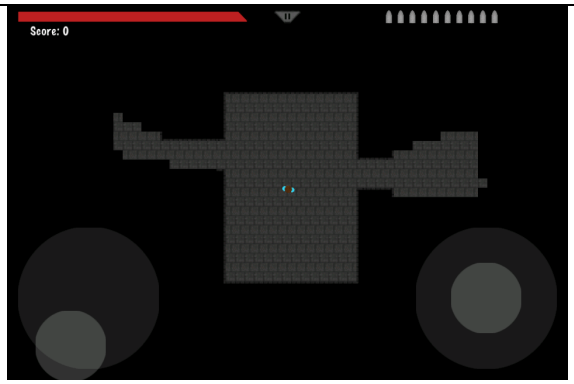
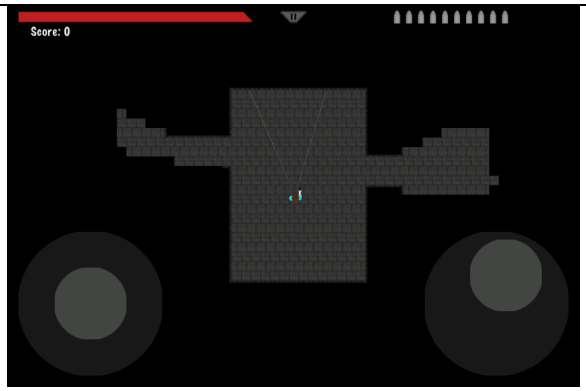
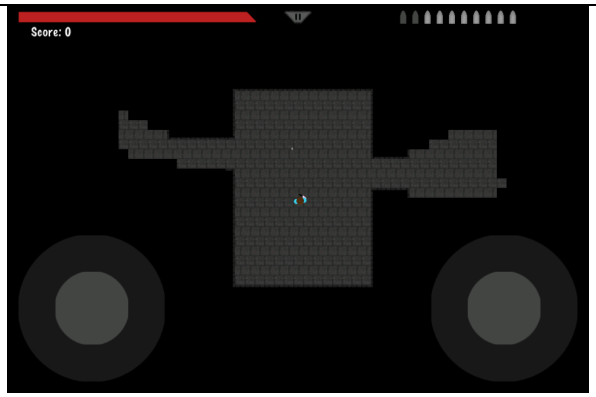
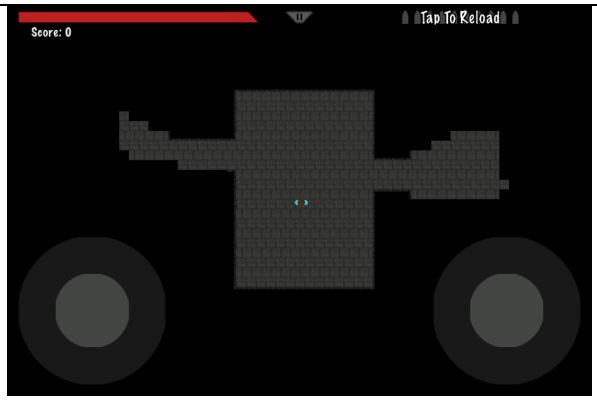
Test Table

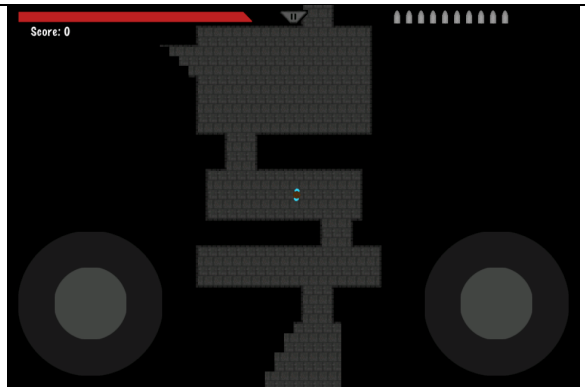
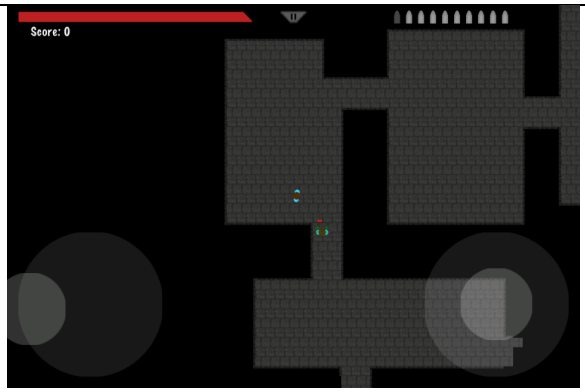
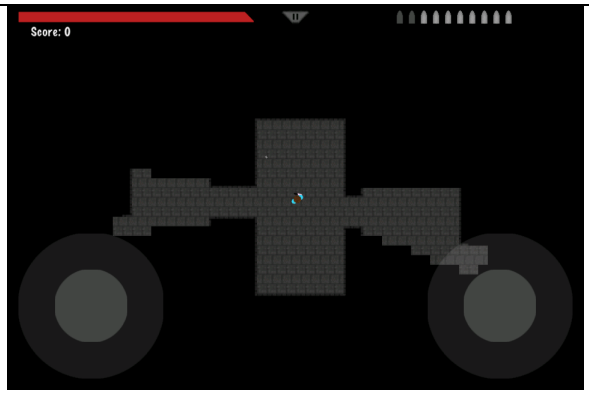
Below is a sample of the tests that I performed on the application. The nature of my application having discrete user interface elements and the control I have over what inputs are available to the user means there are very few possible erroneous inputs.

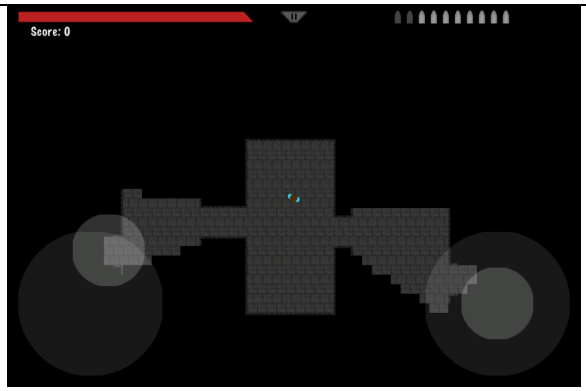
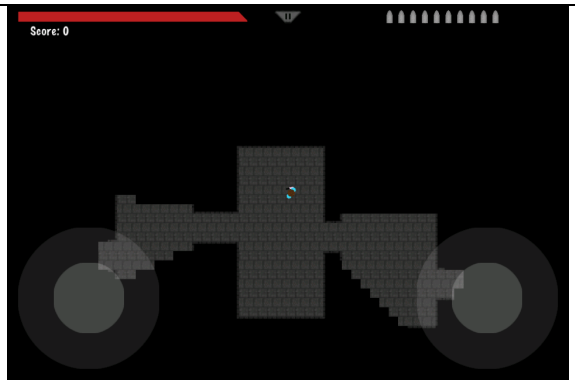
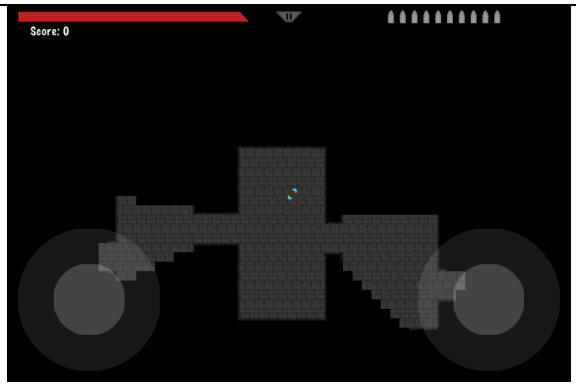
Test Number	Description	Data Type	Expected Result	Pass/Fail	Screen-shot
1	Dragging the movement stick	typical	Moves the player in the correct direction	Pass	1
2	Dragging the aiming stick	typical	Points the player in the correct direction	Pass	2
3	Releasing the aiming stick with ammunition	typical	Fires a bullet in the direction the player is facing	Pass	3
4	Releasing the aiming stick with no ammunition	typical	Does not fire a bullet	Pass	4
5	Map generates correctly	typical	Map is traversable and random	Pass	5
6	Enemies execute pathfinding correctly	typical	Enemies follow the fastest path to the player and do not travel through walls	Pass	6
7	Not typing in a name on the game over screen and tapping ‘done’	erroneous	Highscores do not update, error message shows	Pass	7
8	Double tapping the aim joystick	extreme	Only one bullet is fired	Pass	8

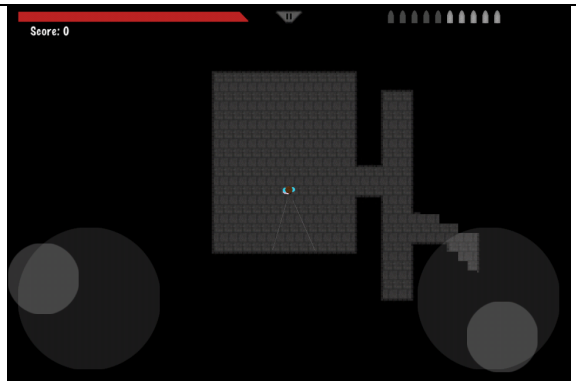
9	Dragging the joysticks beyond their zones	erroneous	Joysticks are moved to the appropriate extreme of their zone	Pass	9
10	Double tapping the reload button	extreme	Player only reloads once	Pass	10
11	Tapping the menu navigation buttons	typical	Application navigates through the specified menus	Pass	11
12	Quitting a game, closing the application and reopening the application	typical	State of the game is recreated as it was when the user quit the game.	Pass	12
13	Using both joysticks at the same time	typical	The player moves while aiming in the correct directions	Pass	13

Annotated Test Run Screenshots

Test Number Cross Reference	Screenshot	Comment
1		The player is moving in the direction the left joystick is dragged.
2		The player is aiming in the direction the right joystick is dragged.
3		The bullet has been fired in the direction the right joystick was pointing when the thumb was released.
4		After releasing my thumb on the right joystick no bullet was fired.

5		The map is random and has been traversable 100% of the time tested.
6		The enemy is walking round the corner, taking the fastest route possible.
7		No name was entered and the shown error message appeared. The highscores were not updated.
8		The second tap on the right joystick did not do anything and the game did not crash.

9		The joystick moved to the extreme position it could take within it's zone such that the direction remained consistent.
10		The player only reloads once and the game does not crash.
11		All the navigation buttons functioned as they were intended
12		The state of the game when loaded was exactly as it was when I quit.

13		Both joysticks acted independently, with the player moving upwards and left while aiming down.

Evaluation

Objectives Achieved

	Objective	Met	Comment
1	A main menu system containing settings.	Yes	There is an easily navigable set of menus with settings, help and high-scores.
2	An in-game menu allowing people to pause and save their game.	Yes	The user can tap a button to pause the game and save their progress then exit the application, or change settings. Progress is saved even after the user closes the application so they can resume their game later.
3	A system of procedurally generating maps.	Yes	When 'new game' is selected a random map is created that is feasible to complete and interesting for the player.
4	Joysticks that interpret touches on the screen as directions and magnitudes.	Yes	The joysticks break down touches on the screen into directions and magnitudes away from their central points.
5	Aiming and movement using the joysticks.	Yes	The player is rotated and moved through dragging the joysticks around.
6	Enemies with intelligent pathfinding and detection mechanics allowing evasion by the player.	Yes	Enemies navigate through the rooms by the optimum path to reach the player and begin chasing the player if he walks within their cone of vision.
7	A line of sight feature shading out areas not explored yet.	Yes	As the player moves through the map visible regions are revealed while regions not yet seen are shrouded in darkness.
8	Runtime created bullet objects to be fired by the player and the enemies.	Yes	The player can shoot bullets and these bullets are each objects initialised upon being fired.
9	Collision detection to prevent the player from walking through walls and to stop bullet objects from passing through walls.	Yes	The player cannot walk through walls and bullets are deallocated upon colliding with walls.
10	Animations, for example seeing the player and enemies aim, reload and walk.	Yes	Animations are included for aiming, shooting and reloading.
11	There should be no noticeable frame rate drops.	Yes	Frame rates were constant at 30 frames per second.

User Feedback

I distributed feedback forms to the same users I identified and questioned in the analysis section. The completed forms can be found in appendix C.

Analysis of User Feedback

Overall the feedback suggests I achieved my numbered objectives and succeeded in creating a game to entertain people for short periods of time in their busy lifestyles. The users also unanimously agreed that the installation was very simple.

The criticisms and suggestions were most helpful from these forms. It appears there are a few areas for improvement, such as making the shadowcasting slightly smoother, widening corridors to make it easier to avoid walking into corners and making the player move faster.

The extensions suggested by the users involved cooperative play so you can play with friends, more game modes, greater enemy and texture variety, and more weapons. These extensions would be possible to implement given more time than is available for the A level computing project.

Possible Extensions

To extend this project, I could increase the longevity of the game by including a variety of enemies with interesting mechanics such as being able to shoot back at the player. This would mean the user would remain interested in the game for a longer amount of time and could be achieved through editing the pathfinding algorithm for each of the new enemies.

The variety of environments could be improved through the addition of new tiles and scenery. This would also keep the user interested in the game for a longer period and attract more people to download it.

A global leader-board would be a potential extension, enabling people to compare their scores with other people on different devices. I could achieve this through using the Game Centre API, meaning I would not need to host a server for the leader-board, saving me money and time in setting up and maintaining it.

Given a large amount of time, I could include a story section, with pre-set missions and maps instead of randomly generated maps. This could be a separate section of the application and would recycle most of the code from the existing game. I would need to manually create the maps required, which I could do using the Tiled application, and program the new functionality such as checkpoints and goals. The checkpoints and goals could be saved as objects on the TMX tile map and code could be included to recognize proximity to these objects.

Appendix B - Program Code

[For this exemplar only a sample of the code is provided - the original document had all of the code written by the student]

EXEMPLAR CONTENTS:

```
//
//  AppDelegate.h
//  ALevelProject
//
//  Created by Daniel Ellis.
//  Copyright Daniel Ellis 2014. All rights reserved.
//

#import <UIKit/UIKit.h>
#import "cocos2d.h"

@interface AppController : NSObject <UIApplicationDelegate, CCDirectorDelegate
    >
{
    UIWindow *window_;
    UINavigationController *navController_;

    CCDirectorIOS   *__unsafe_unretained director_;                          //
        weak ref

}

@property (nonatomic, strong) UIWindow *window;
@property (readonly) UINavigationController *navController;
@property (unsafe_unretained, readonly) CCDirectorIOS *director;

@end
```

```
//
//  AppDelegate.mm
//  ALevelProject
//
//  Created by Daniel Ellis.
//  Copyright Daniel Ellis 2014. All rights reserved.
//

#import "cocos2d.h"

#import "AppDelegate.h"
#import "IntroLayer.h"
#import "Engine.h"
#import "GUIMainController.h"

@implementation AppController

@synthesize window=window_, navController=navController_, director=director_;

- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // Create the main window
    window_ = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];


    // Create an CCGLView with a RGB565 color buffer, and a depth buffer of 0-
        bits
    CCGLView *glView = [CCGLView viewWithFrame:[window_ bounds]
                                   pixelFormat:kEAGLColorFormatRGB565   //
                                       kEAGLColorFormatRGBA8
                                   depthFormat:0    //GL_DEPTH_COMPONENT24_OES
                            preserveBackbuffer:NO
                                    sharegroup:nil
                                 multiSampling:NO
                               numberOfSamples:0];

    // Multiple Touches enabled
    [glView setMultipleTouchEnabled:YES];

    director_ = (CCDirectorIOS*) [CCDirector sharedDirector];

    //director_.wantsFullScreenLayout = YES;
    [[UIApplication sharedApplication] setStatusBarHidden:YES withAnimation:NO
        ];

    // Display FSP and SPF
    //[director_ setDisplayStats:YES];

    // set FPS at 30
    [director_ setAnimationInterval:1.0/30];

    // attach the openglView to the director
    [director_ setView:glView];

    // for rotation and other messages
    [director_ setDelegate:self];

    // 2D projection
    [director_ setProjection:kCCDirectorProjection2D];
```

```
    //  [director setProjection:kCCDirectorProjection3D];

    // Enables High Res mode (Retina Display) on iPhone 4 and maintains low
        res on all other devices
    if( ! [director_ enableRetinaDisplay:YES] )
        CCLOG(@"Retina Display Not supported");

    // Default texture format for PNG/BMP/TIFF/JPEG/GIF images
    // It can be RGBA8888, RGBA4444, RGB5_A1, RGB565
    // You can change anytime.
    [CCTexture2D setDefaultAlphaPixelFormat:kCCTexture2DPixelFormat_RGBA8888];

    // If the 1st suffix is not found and if fallback is enabled then fallback
        suffixes are going to searched. If none is found, it will try with the
        name without suffix.
    // On iPad HD  : "-ipadhd", "-ipad",  "-hd"
    // On iPad     : "-ipad", "-hd"
    // On iPhone HD: "-hd"
    CCFileUtils *sharedFileUtils = [CCFileUtils sharedFileUtils];
    [sharedFileUtils setEnableFallbackSuffixes:NO];             // Default:
        NO. No fallback suffixes are going to be used
    [sharedFileUtils setiPhoneRetinaDisplaySuffix:@"-hd"];      // Default on
        iPhone RetinaDisplay is "-hd"
    [sharedFileUtils setiPadSuffix:@"-ipad"];                   // Default on
        iPad is "ipad"
    [sharedFileUtils setiPadRetinaDisplaySuffix:@"-ipadhd"];    // Default on
        iPad RetinaDisplay is "-ipadhd"

    // Assume that PVR images have premultiplied alpha
    [CCTexture2D PVRImagesHavePremultipliedAlpha:YES];

    // and add the scene to the stack. The director will run it when it
        automatically when the view is displayed.
    [director_ pushScene: [IntroLayer scene]];


    // Create a Navigation Controller with the Director
    navController_ = [[UINavigationController alloc]
        initWithRootViewController:director_];
    navController_.navigationBarHidden = YES;

    // set the Navigation Controller as the root view controller
//  [window_ addSubview:navController_.view];   // Generates flicker.
    [window_ setRootViewController:navController_];

    // make main window visible
    [window_ makeKeyAndVisible];


    // load data from property list
    NSString *errorDesc = nil;
    NSPropertyListFormat format;
    NSString *plistPath;
    NSString *rootPath = [NSSearchPathForDirectoriesInDomains
        (NSDocumentDirectory,
                                                          NSUserDomainMask
                                                    , YES) objectAtIndex:0];
    plistPath = [rootPath stringByAppendingPathComponent:@"Data.plist"];
    if (![[NSFileManager defaultManager] fileExistsAtPath:plistPath]) {
        plistPath = [[NSBundle mainBundle] pathForResource:@"Data"
```

```
            ofType:@"plist"];
    }
    NSData *plistXML = [[NSFileManager defaultManager] contentsAtPath:
        plistPath];
    NSDictionary *temp = (NSDictionary *)[NSPropertyListSerialization
                                          propertyListFromData:plistXML
                                          mutabilityOption:
                                              NSPropertyListMutableContainersAn
                                              dLeaves
                                          format:&format
                                          errorDescription:&errorDesc];
    if (!temp) {
        NSLog(@"Error reading plist: %@, format: %d", errorDesc, format);
    }
    [[Engine sharedInstance] setAlive: [temp objectForKey:@"alive"]];
    [[Engine sharedInstance] setPlayerX: [temp objectForKey:@"playerx"]];
    [[Engine sharedInstance] setPlayerY: [temp objectForKey:@"playery"]];
    [[Engine sharedInstance] setGoreOn: [temp objectForKey:@"gore"]];
    [[Engine sharedInstance] setMusicOn: [temp objectForKey:@"music"]];
    [[Engine sharedInstance] setSfxOn: [temp objectForKey:@"sfx"]];
    [[Engine sharedInstance] setHealth: [temp objectForKey:@"health"]];
    [[Engine sharedInstance] setAmmo: [temp objectForKey:@"ammo"]];
    [[Engine sharedInstance] setCurrentScore: [temp objectForKey:@"score"]];
    [[Engine sharedInstance] setCurrentLevel: [temp objectForKey:@"level"]];
    [[Engine sharedInstance] setSaved: [temp objectForKey:@"saved"]];

    NSMutableArray *array = [temp objectForKey:@"highscores"];
    NSMutableArray *array2 = [temp objectForKey:@"highscorenames"];

    for (int x = 0; x <= 4; x++)
    {
        NSNumber *number = [array objectAtIndex:x];
        [[[Engine sharedInstance] highScores] replaceObjectAtIndex:x
            withObject:number];
        number = [array2 objectAtIndex:x];
        [[[Engine sharedInstance] highScoreNames] replaceObjectAtIndex:x
            withObject:number];
    }

    return YES;
}

// Supported orientations: Landscape. Customize it for your own needs
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)
    interfaceOrientation
{
    return UIInterfaceOrientationIsLandscape(interfaceOrientation);
}


// getting a call, pause the game
-(void) applicationWillResignActive:(UIApplication *)application
{
    if( [navController_ visibleViewController] == director_ )
        [director_ pause];
}

// call got rejected
-(void) applicationDidBecomeActive:(UIApplication *)application
{
```

```
    if( [navController_ visibleViewController] == director_ )
        [director_ resume];
}

-(void) applicationDidEnterBackground:(UIApplication*)application
{
    //save data to property list
    if( [navController_ visibleViewController] == director_ )
        [director_ stopAnimation];

    NSString *error;
    NSString *rootPath = [NSSearchPathForDirectoriesInDomains
        (NSDocumentDirectory, NSUserDomainMask, YES) objectAtIndex:0];
    NSString *plistPath = [rootPath
        stringByAppendingPathComponent:@"Data.plist"];


    NSDictionary *plistDict = [NSDictionary dictionaryWithObjects:[NSArray
        arrayWithObjects:[[Engine sharedInstance] playerX],[[Engine
        sharedInstance] playerY],[[Engine sharedInstance] highScoreNames],
        [[Engine sharedInstance] highScores],[[Engine sharedInstance] alive],
        [[Engine sharedInstance] goreOn], [[Engine sharedInstance] musicOn],
        [[Engine sharedInstance] sfxOn], [[Engine sharedInstance] currentScore]
        , [[Engine sharedInstance] health], [[Engine sharedInstance] ammo],
        [[Engine sharedInstance] currentLevel], [[Engine sharedInstance] saved]
        , nil] forKeys:[NSArray arrayWithObjects:@"playerx", @"playery",
        @"highscorenames",
        @"highscores",@"alive",@"gore",@"music",@"sfx",@"score",@"health",@"amm
        o",@"level",@"saved", nil]];

    NSData *plistData = [NSPropertyListSerialization dataFromPropertyList:
        plistDict format:NSPropertyListXMLFormat_v1_0 errorDescription:&error];

    if(plistData)
    {
        [plistData writeToFile:plistPath atomically:YES];
    }

}

-(void) applicationWillEnterForeground:(UIApplication*)application
{
    if( [navController_ visibleViewController] == director_ )
        [director_ startAnimation];
}

// application will be killed
- (void)applicationWillTerminate:(UIApplication *)application
{
    CC_DIRECTOR_END();
}

// purge memory
- (void)applicationDidReceiveMemoryWarning:(UIApplication *)application
{
    [[CCDirector sharedDirector] purgeCachedData];
}

// next delta time will be zero
-(void) applicationSignificantTimeChange:(UIApplication *)application
```

```
{
    [[CCDirector sharedDirector] setNextDeltaTimeZero:YES];
}

@end
```

```
//
//  BSPNode.h
//  ALevelProject
//
//  Created by Daniel Ellis.
//  Copyright 2014 Daniel Ellis. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "cocos2d.h"

@interface BSPNode : NSObject {

}

@property (strong) NSNumber *topRightx;
@property (strong) NSNumber *topRighty;
@property (strong) NSNumber *botLeftx;
@property (strong) NSNumber *botLefty;
@property (strong) BSPNode *leftNode;
@property (strong) BSPNode *rightNode;
@property (strong) NSNumber *vertical;
@property (strong) NSNumber *split;

@end
```

```
//
//  BSPNode.m
//  ALevelProject
//
//  Created by Daniel Ellis.
//  Copyright 2014 Daniel Ellis. All rights reserved.
//

#import "BSPNode.h"


@implementation BSPNode

@synthesize botLeftx;
@synthesize botLefty;
@synthesize topRightx;
@synthesize topRighty;
@synthesize leftNode;
@synthesize rightNode;
@synthesize vertical;
@synthesize split;

@end
```

```
//
//  Bullet.h
//  ALevelProject
//
//  Created by Daniel Ellis.
//  Copyright 2014 Daniel Ellis. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "cocos2d.h"

@interface Bullet : CCSprite {

}

@property (strong) NSNumber *damage;
@property (strong) NSNumber *direction;
@property (strong) NSNumber *dead;
- (void) setupWithDamage:(int)damage direction:(double)direction;
- (CGPoint) futurePos;
- (void) updatePosition:(CGPoint)newPos;
- (void) stopped;

@end
```

```
//
//  Bullet.m
//  ALevelProject
//
//  Created by Daniel Ellis.
//  Copyright 2014 Daniel Ellis. All rights reserved.
//

#import "Bullet.h"

@interface Bullet()

@property (strong) NSNumber *vX;
@property (strong) NSNumber *vY;


@end

@implementation Bullet

- (void)setupWithDamage:(int)damage direction:(double)direction
{
    // set up the bullet's properties as it is fired
    _damage = [NSNumber numberWithDouble:damage];
    _direction = [NSNumber numberWithDouble:direction];
    self.rotation = _direction.integerValue;
    _vX = [NSNumber numberWithDouble:((damage/25) + 2) * sin(direction * 2*
        3.141592654/360)];
    _vY = [NSNumber numberWithDouble:((damage/25) + 2) * cos(direction * 2*
        3.141592654/360)];

    double tempX = cos(atan(_vY.doubleValue/_vX.doubleValue)) * 16;
    double tempY = sin(atan(_vY.doubleValue/_vX.doubleValue)) * 16;
    if (_vX.doubleValue < 0)
    {
        tempX = -tempX;
        tempY = -tempY;
    }

    [self setPosition:ccp(self.position.x + tempX, self.position.y + tempY)];
    _dead = [NSNumber numberWithBool:NO];
}

- (void) updatePosition:(CGPoint)newPos
{
    //update the position of the bullet
    self.position = newPos;
}

- (void) stopped
{
    //stop the bullet
    _vX = [NSNumber numberWithInt:0];
    _vY = [NSNumber numberWithInt:0];
    _dead = [NSNumber numberWithBool:YES];
}

- (CGPoint) futurePos
{
    //return the position of the bullet when it is moved
```

```
    return ccp(self.position.x + _vX.doubleValue, self.position.y + _vY.
        doubleValue);
}

@end
```

```
//
//  Enemy.h
//  ALevelProject
//
//  Created by Daniel Ellis.
//  Copyright 2014 Daniel Ellis. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "cocos2d.h"

@interface Enemy : CCSprite {

}

- (void) initWithHealth:(int)health Damage:(int)damage;
- (BOOL) hitWithDamageDead:(double)damage Direction:(double)direction Map:
    (CCTMXTiledMap*)map Layer:(CCTMXLayer*)background;
- (void) movement:(int)endX :(int)endY Layer:(CCTMXLayer*)background Map:
    (CCTMXTiledMap*)map;
- (BOOL) checkForAttack;
- (CCProgressTimer*) healthDisplayReference;
@property (strong) NSNumber *dead;
@property (strong) NSNumber *health;
@property (strong) NSNumber *activated;
@end
```

```
//
//  Enemy.m
//  ALevelProject
//
//  Created by Daniel Ellis.
//  Copyright 2014 Daniel Ellis. All rights reserved.
//

#import "Enemy.h"
#import "PathfindingNode.h"
#import "Engine.h"

@interface Enemy()

@property (strong) NSNumber *damage;
@property (strong) NSNumber *hunting;
@property (strong) NSNumber *huntDestX;
@property (strong) NSNumber *huntDestY;
@property (strong) NSNumber *huntDestFracX;
@property (strong) NSNumber *huntDestFracY;
@property (strong) NSNumber *nextHuntDestX;
@property (strong) NSNumber *nextHuntDestY;
@property (strong) NSNumber *nextHuntDestFracX;
@property (strong) NSNumber *nextHuntDestFracY;
@property (strong) NSNumber *processing;
@property (strong) NSNumber *nextSimple;
@property (strong) NSNumber *nextProcessed;
@property (strong) NSNumber *endX;
@property (strong) NSNumber *endY;
@property (strong) NSNumber *startX;
@property (strong) NSNumber *startY;
@property (unsafe_unretained) CCTMXLayer *background;
@property (unsafe_unretained) CCTMXTiledMap *map;
@property (strong) NSNumber *attackCounter;
@property (strong) CCProgressTimer *healthDisplay;

@end

@implementation Enemy

-(BOOL)spaceIsBlocked:(int)x :(int)y Layer:(CCTMXLayer*)background Map:
    (CCTMXTiledMap*)map
{
    //return whether the vertex specified is blocked by any of the 4 tiles
    bool col1 = [[map propertiesForGID:[background tileGIDAt:ccp(x - 1, map.
        mapSize.height - y - 1)]][@"Collidable"] isEqualToString:@"True"];
    bool col2 = [[map propertiesForGID:[background tileGIDAt:ccp(x, map.
        mapSize.height - y - 1)]][@"Collidable"] isEqualToString:@"True"];
    bool col3 = [[map propertiesForGID:[background tileGIDAt:ccp(x - 1, map.
        mapSize.height - y)]][@"Collidable"] isEqualToString:@"True"];
    bool col4 = [[map propertiesForGID:[background tileGIDAt:ccp(x, map.
        mapSize.height - y)]][@"Collidable"] isEqualToString:@"True"];
    if(col1 || col2 || col3 || col4)
    {
        return YES;
    }
    else
    {
        return NO;
    }
```

```
}

-(PathfindingNode*)nodeInArray:(NSMutableArray*)a withX:(int)x Y:(int)y
{
    //return the node with coordinates (x,y) in the inputted array
    NSEnumerator *e = [a objectEnumerator];
    PathfindingNode *n;

    while((n = [e nextObject]))
    {
        if((n.x.intValue == x) && (n.y.intValue == y))
        {
            return n;
        }
    }

    return nil;
}
-(PathfindingNode*)lowestCostNodeInArray:(NSMutableArray*)a
{
    //Finds the node with the lowest cost
    PathfindingNode *n, *lowest;
    lowest = nil;
    NSEnumerator *e = [a objectEnumerator];

    while((n = [e nextObject]))
    {
        if(lowest == nil)
        {
            lowest = n;
        }
        else
        {
            if(n.gCost.floatValue < lowest.gCost.floatValue)
            {
                lowest = n;
            }
        }
    }
    return lowest;
}

- (void) movement:(int)endX :(int)endY Layer:(CCTMXLayer*)background Map:
    (CCTMXTiledMap*)map
{
    //determine whether and how the enemy should move to the player
    if (_dead.boolValue == false)
    {
        if (_activated.boolValue == true)
        {
            if (_hunting.boolValue == false)
            {
                if (_processing.boolValue == true)
                {
                    return;
                }
                else if (_nextProcessed.boolValue == true)
                {
                    if (_nextSimple.boolValue == true)
                    {
```

```
                    _nextSimple = [NSNumber numberWithBool:false];
                    _nextProcessed = [NSNumber numberWithBool:false];
                    self.rotation = atan2f(endX - self.position.x, endY -
                        self.position.y) * (360/(2*3.141592654));
                    if (abs(endX - self.position.x) <= 14 && abs(endY -
                        self.position.y) <= 14)
                    {
                        //attack
                    }
                    else
                    {
                        [self setPosition:ccp(self.position.x + (sin(self.
                            rotation * 2*3.141592654/360)), self.position.y
                            + (cos(self.rotation * 2*3.141592654/360)))];
                    }
                }
                else
                {
                    _huntDestX = [NSNumber numberWithInt:_nextHuntDestX.
                        intValue];
                    _huntDestY = [NSNumber numberWithInt:_nextHuntDestY.
                        intValue];
                    _huntDestFracX = [NSNumber numberWithDouble:
                        _nextHuntDestFracX.doubleValue];
                    _huntDestFracY = [NSNumber numberWithDouble:
                        _nextHuntDestFracY.doubleValue];
                    _hunting = [NSNumber numberWithBool:true];
                    _nextProcessed = [NSNumber numberWithBool:false];
                    int angle = (atan2f(_huntDestX.intValue - self.
                        position.x, _huntDestY.intValue - self.position.y)
                        * (360/(2*3.141592654)));
                    if ((_huntDestX.intValue<0 && _huntDestY.intValue < 0)
                        || (_huntDestX.intValue>=0 && _huntDestY.intValue<0
                        ))
                    {
                        angle += 180;
                    }
                    self.rotation = angle;
                }
            }
            else
            {
                _endX = [NSNumber numberWithInt:endX];
                _endY =[NSNumber numberWithInt:endY];
                _startX = [NSNumber numberWithInt:self.position.x];
                _startY = [NSNumber numberWithInt:self.position.y];

                _background = background;
                _map = map;
                [self performSelectorInBackground:@selector
                    (testMethodCurrentMove) withObject:nil];
            }
        }
        else if (_processing.boolValue == false)
        {
            if (_nextProcessed.boolValue == false)
            {
                _endX = [NSNumber numberWithInt:endX];
                _endY = [NSNumber numberWithInt:endY];
                int tempx = _huntDestX.intValue;
```

```
                    int tempy = _huntDestY.intValue;
                    _startX = [NSNumber numberWithInt:tempx];
                    _startY = [NSNumber numberWithInt:tempy];
                    _background = background;
                    _map = map;
                    [self performSelectorInBackground:@selector
                        (testMethodNextMove) withObject:nil];
                }
                [self hunt];
            }
            else
            {
                [self hunt];
            }
        }
        else
        {
            _activated = [NSNumber numberWithBool:[self directLine:ccp(endX,
                endY) Layer:background Map:map]];
        }
    }
}

- (void) pathFinderWithEndX:(int)endX EndY:(int)endY StartX:(int)startX
    StartY:(int)startY Layer:(CCTMXLayer*)background Map:(CCTMXTiledMap*)map
    ProcessingNextMove:(bool)processingNextMove
{
    // find the next vertex for the enemy to move to in order to get to the
        player, using the A* pathfinding method
    _processing = [NSNumber numberWithBool:true];
    BOOL simple = true;
    double fracx = (endX - startX);
    double fracy = (endY - startY);
    fracx /= 50;
    fracy /= 50;
    int x;
    int y;
    int GID;
    NSDictionary *properties;
    NSString *collision;
    //check if the enemy can simply walk directly to the player
    for (int n = 1; n <= 50; n++)
    {
        //line from right of enemy to right of player
        x = (startX + 6 + (n*fracx))/8;
        y = ((map.mapSize.height * map.tileSize.height) - startY - (n*fracy))/
            8;
        GID = [background tileGIDAt:ccp(x,y)];
        properties = [map propertiesForGID:GID];
        collision = properties[@"Collidable"];

        if ([collision isEqualToString:@"True"])
        {
            simple = false;
            break;
        }

        //line from left of enemy to left of player
        x = (startX - 6 + (n*fracx))/8;
        y = ((map.mapSize.height * map.tileSize.height) - startY - (n*fracy))/
```

```
            8;
        GID = [background tileGIDAt:ccp(x,y)];
        properties = [map propertiesForGID:GID];
        collision = properties[@"Collidable"];

        if ([collision isEqualToString:@"True"])
        {
            simple = false;
            break;
        }

        //line from bottom of enemy to bottom of player
        x = (startX + (n*fracx))/8;
        y = ((map.mapSize.height * map.tileSize.height) - startY + 6 - (n*
            fracy))/8;
        GID = [background tileGIDAt:ccp(x,y)];
        properties = [map propertiesForGID:GID];
        collision = properties[@"Collidable"];

        if ([collision isEqualToString:@"True"])
        {
            simple = false;
            break;
        }

        //line from top of enemy to top of player
        x = (startX + (n*fracx))/8;
        y = ((map.mapSize.height * map.tileSize.height) - startY - 6 - (n*
            fracy))/8;
        GID = [background tileGIDAt:ccp(x,y)];
        properties = [map propertiesForGID:GID];
        collision = properties[@"Collidable"];

        if ([collision isEqualToString:@"True"])
        {
            simple = false;
            break;
        }
    }

    if (simple == true)
    {
        //can just walk towards the player
        if (processingNextMove == false)
        {
            self.rotation = atan2f(endX - startX, endY - startY) * (360/(2*
                3.141592654));
            if (abs(endX - startX) <= 16 && abs(endY - startY) <= 16)
            {
                [self attack];
            }
            else
            {
                [self setPosition:ccp(self.position.x + (1.5*sin(self.rotation
                    * 2*3.141592654/360)), self.position.y + (1.5*cos(self.
                    rotation * 2*3.141592654/360)))];
            }
        }
        else
        {
```

```
            _nextSimple = [NSNumber numberWithBool:true];
            _nextProcessed = [NSNumber numberWithBool:true];
        }
        _processing = [NSNumber numberWithBool:false];
    }
    else
    {
        //find path, cannot simply walk towards the player
        if (endX%8 > 4)
        {
            endX /= 8;
            endX += 1;
        }
        else
        {
            endX /= 8;
        }

        if (endY%8 > 4)
        {
            endY /= 8;
            endY += 1;
        }
        else
        {
            endY /= 8;
        }

        int newX,newY;
        int currentX,currentY;
        NSMutableArray *openList, *closedList;

        x = startX;
        x /= 8;
        y = startY;
        y /= 8;

        if (startX%8 > 4)
        {
            x++;
        }
        if (startY%8 > 4)
        {
            y++;
        }

        if (processingNextMove == false)
        {
            self.position = ccp(x*8,y*8);
            startX = x*8;
            startY = y*8;
        }
        else
        {
            startX = x*8;
            startY = y*8;
        }

        if((x == endX) && (y == endY))
        {
```

```
        return;
    }
    openList = [NSMutableArray array];
    closedList = [NSMutableArray array];

    PathfindingNode *currentNode = nil;
    PathfindingNode *adjacentNode = nil;
    PathfindingNode *startNode = [[PathfindingNode alloc] init];
    startNode.x = [NSNumber numberWithInt:x];
    startNode.y = [NSNumber numberWithInt:y];
    startNode.source = nil;
    startNode.fCost = [NSNumber numberWithInt:0];
    startNode.gCost = [NSNumber numberWithFloat:sqrtf(((x - endX)*(x -
        endX)) + ((y - endY)*(y-endY)))];
    [openList addObject: startNode];

    //start algorithm
    while([openList count])
    {
        currentNode = [self lowestCostNodeInArray: openList];
        if((currentNode.x.intValue == endX) && (currentNode.y.intValue ==
            endY))
        {
            //reached the player
            currentNode = currentNode.source;
            if (currentNode.source != nil)
            {
                while(currentNode.source.source != nil)
                {
                    currentNode = currentNode.source;
                }
            }

            if (processingNextMove == false)
            {
                //running on the main thread
                _huntDestX = [NSNumber numberWithInt:(currentNode.x.
                    intValue)*8];
                _huntDestY = [NSNumber numberWithInt:(currentNode.y.
                    intValue)*8];

                int angle = (atan2f(_huntDestX.intValue - self.position.x,
                    _huntDestY.intValue - self.position.y) * (360/(2*
                    3.141592654)));
                if ((_huntDestX.intValue<0 && _huntDestY.intValue < 0) ||
                    (_huntDestX.intValue>=0 && _huntDestY.intValue<0))
                {
                    angle += 180;
                }

                _huntDestFracX = [NSNumber numberWithDouble:1.5*sin(angle*
                    ((2*3.141592654)/360))];
                _huntDestFracY = [NSNumber numberWithDouble:1.5*cos(angle*
                    ((2*3.141592654)/360))];
                if ((_huntDestX.intValue < 0 && _huntDestY.intValue < 0) |
                    | (_huntDestX.intValue >= 0 && _huntDestY.intValue < 0)
                    )
                {
                    _huntDestFracX = [NSNumber numberWithDouble:-
                        _huntDestFracX.doubleValue];
```

```
                _huntDestFracY = [NSNumber numberWithDouble:-
                    _huntDestFracY.doubleValue];
            }

            _hunting = [NSNumber numberWithBool:true];
            self.rotation = angle;
        }
        else
        {
            //running on another thread
            _nextHuntDestX = [NSNumber numberWithInt:currentNode.x.
                intValue*8];
            _nextHuntDestY = [NSNumber numberWithInt:currentNode.y.
                intValue*8];

            int angle = (atan2f(_nextHuntDestX.intValue - startX,
                _nextHuntDestY.intValue - startY) * (360/(2*3.141592654
                )));
            if ((_nextHuntDestX.intValue<0 && _nextHuntDestY.intValue
                < 0) || (_nextHuntDestX.intValue>=0 && _nextHuntDestY.
                intValue<0))
            {
                angle += 180;
            }

            _nextHuntDestFracX = [NSNumber numberWithDouble:1.5*sin
                (angle*((2*3.141592654)/360))];
            _nextHuntDestFracY = [NSNumber numberWithDouble:1.5*cos
                (angle*((2*3.141592654)/360))];
            if ((_nextHuntDestX.doubleValue < 0 && _nextHuntDestY.
                doubleValue < 0) || (_nextHuntDestX.doubleValue >= 0 &&
                _nextHuntDestY.doubleValue < 0))
            {
                _nextHuntDestFracX = [NSNumber numberWithDouble:-
                    _nextHuntDestFracX.doubleValue];
                _nextHuntDestFracY = [NSNumber numberWithDouble:-
                    _nextHuntDestFracY.doubleValue];
            }
            _nextProcessed = [NSNumber numberWithBool:true];
            _nextSimple = [NSNumber numberWithBool:false];
        }
        _processing = [NSNumber numberWithBool:false];
        return;
    }
    else
    {
        //A* section
        [closedList addObject: currentNode];
        [openList removeObject: currentNode];

        currentX = currentNode.x.intValue;
        currentY = currentNode.y.intValue;

        for(y=-1;y<=1;y++)
        {
            newY = currentY+y;
            for(x=-1;x<=1;x++)
            {
                newX = currentX+x;
                if(![self nodeInArray: openList withX: newX Y:newY])
```

```
                            {
                                if(![self nodeInArray: closedList withX: newX Y:
                                    newY])
                                {
                                    if((![self spaceIsBlocked: newX :newY Layer:
                                        background Map:map]) || ((newX == endX) &&
                                        (newY == endX)))
                                    {
                                        adjacentNode = [[PathfindingNode alloc]
                                            init];
                                        adjacentNode.x = [NSNumber numberWithInt:
                                            newX];
                                        adjacentNode.y = [NSNumber numberWithInt:
                                            newY];
                                        adjacentNode.source = currentNode;
                                        adjacentNode.fCost = [NSNumber
                                            numberWithFloat:currentNode.fCost.
                                            floatValue + sqrtf(((newX - currentX)*
                                            (newX - currentX)) + ((newY - currentY)
                                            *(newY-currentY)))];
                                        adjacentNode.gCost = [NSNumber
                                            numberWithFloat:adjacentNode.fCost.
                                            floatValue + sqrtf(((newX - endX)*(newX
                                            - endX)) + ((newY - endY)*(newY-endY)))
                                            ];

                                        [openList addObject: adjacentNode];
                                    }
                                }
                            }
                        }
                    }
                }
            }
        _processing = [NSNumber numberWithBool:false];
    }
}

- (void) hunt
{
    //move the enemy towards the vertex specified by the A* algorithm
    float checkInt = ((_huntDestX.intValue - self.position.x)*(_huntDestX.
        intValue - self.position.x));

    checkInt += ((_huntDestY.intValue - self.position.y)*(_huntDestY.intValue
        - self.position.y));

    if (checkInt >= 1)
    {
        self.position = ccp(self.position.x + _huntDestFracX.doubleValue, self
            .position.y + _huntDestFracY.doubleValue);
    }
    else
    {
        self.position = ccp(_huntDestX.intValue,_huntDestY.intValue);
        _hunting = [NSNumber numberWithBool:false];
    }
}

- (BOOL) directLine:(CGPoint)playerPos Layer:(CCTMXLayer*)background Map:
```

```
    (CCTMXTiledMap*) map
{
    // check if the enemy can see the player
    if  (((self.position.x - playerPos.x)*(self.position.x- playerPos.x)) +
        ((self.position.y- playerPos.y) * (self.position.y- playerPos.y)) <=
        360000)
    {
        int dx = playerPos.x - self.position.x;
        int dy = playerPos.y - self.position.y;
        float direction = atan2f(abs(dy), abs(dx)) * (360/(2*3.141592654));
        if (dx < 0)
        {
            if (dy < 0)
            {
                direction = 270 - direction;
            }
            else
            {
                direction = 270 + direction;
            }
        }
        else if (dy > 0)
        {
            direction = 90 - direction;
        }
        else
        {
            direction = 90 + direction;
        }

        if (abs(direction - self.rotation) > 70 && MIN(direction,self.rotation
            ) + 70 % 360 < MAX(direction,self.rotation))
        {
            return false;
        }
        double fracx = (playerPos.x - self.position.x)/100;
        double fracy = (playerPos.y - self.position.y)/100;
        int x;
        int y;
        int GID;
        NSDictionary *properties;
        NSString *collision;
        for (int n = 1; n <= 100; n++)
        {
            x = (self.position.x + (n*fracx))/8;
            y = ((map.mapSize.height * map.tileSize.height) - self.position.y
                - (n*fracy))/8;
            GID = [background tileGIDAt:ccp(x,y)];
            properties = [map propertiesForGID:GID];
            collision = properties[@"Collidable"];

            if ([collision isEqualToString:@"True"])
            {
                return false;
            }
        }
        return true;
    }
    return false;
}
```

```
- (void) initWithHealth:(int)health Damage:(int)damage
{
    //initialise the enemy
    _health = [NSNumber numberWithInt:health];
    _damage = [NSNumber numberWithInt:damage];
    _dead = [NSNumber numberWithBool:false];
    _activated = [NSNumber numberWithBool:false];
    _hunting = [NSNumber numberWithBool:false];
    _processing = [NSNumber numberWithBool:false];
    _nextSimple = [NSNumber numberWithBool:false];
    _nextProcessed = [NSNumber numberWithBool:false];
    _attackCounter = [NSNumber numberWithInt:0];

    _healthDisplay = [CCProgressTimer progressWithSprite:[CCSprite
        spriteWithFile:@"EnemyHealth.png"]];
    _healthDisplay.type = kCCProgressTimerTypeBar;
    _healthDisplay.midpoint = ccp(0,0);
    _healthDisplay.barChangeRate = ccp(1,0);
    _healthDisplay.percentage = _health.integerValue;
    _healthDisplay.position = ccp(self.position.x,self.position.y + 10);
}

- (BOOL) hitWithDamageDead:(double)damage Direction:(double)direction Map:
    (CCTMXTiledMap*)map Layer:(CCTMXLayer*)background
{
    //enemy has been hit with a bullet, deduct health, check for death and
        knock backwards
    if (!_activated.boolValue)
    {
        _activated = [NSNumber numberWithBool:true];
    }
    _health = [NSNumber numberWithInt:_health.intValue - damage];
    if (_health.intValue <= 0)
    {
        _dead = [NSNumber numberWithBool:true];
        return true;
    }
    else
    {
        if([[map propertiesForGID:[background tileGIDAt:ccp((self.position.x +
            (8*sin(direction*((2*3.141592654)/360))))/8,((map.mapSize.height *
            map.tileSize.height) - self.position.y - (8*cos(direction*((2*
            3.141592654)/360))))/8)]][@"Collidable"] isEqualToString:@"True"])
        {
            [self setPosition:ccp(self.position.x + ((damage/12.5) * sin
                (direction * 2*3.141592654/360)), self.position.y + ((damage/
                12.5) * cos(direction * 2*3.141592654/360)))];
            _hunting = [NSNumber numberWithBool:false];
            _nextProcessed = [NSNumber numberWithBool:false];
            _nextSimple = [NSNumber numberWithBool:false];
        }
    }
    return false;
}


- (void) testMethodNextMove
{
    //execute the pathfinding algorithm, starting at the point being moved to
```

```
        by the enemy
    int tempX = _startX.intValue;
    int tempY = _startY.intValue;
    [self pathFinderWithEndX:_endX.intValue EndY:_endY.intValue StartX:tempX
        StartY:tempY Layer:_background Map:_map ProcessingNextMove:true];
}

- (void) testMethodCurrentMove
{
    //execute the pathfinding algorithm, starting at the current position of
        the enemy
    [self pathFinderWithEndX:_endX.intValue EndY:_endY.intValue StartX:_startX
        .intValue StartY:_startY.intValue Layer:_background Map:_map
        ProcessingNextMove:false];
}

- (void) attack
{
    //delay the frequency of attacks on the player using a counter
    if (_attackCounter.intValue != 4)
    {
        _attackCounter = [NSNumber numberWithInt:_attackCounter.intValue + 1];
    }
}

- (BOOL) checkForAttack
{
    //attack the player, using the delay counter
    if (_attackCounter.intValue == 4)
    {
        _attackCounter = [NSNumber numberWithInt:0];
        return true;
    }
    return false;
}

- (CCProgressTimer*) healthDisplayReference
{
    //return the address of the health bar object above the enemy
    return _healthDisplay;
}

@end
```

```
//
//  Engine.h
//  ALevelProject
//
//  Created by Daniel Ellis.
//  Copyright 2014 Daniel Ellis. All rights reserved.
//

#import <Foundation/Foundation.h>

@interface Engine : NSObject
{

}

@property (copy, nonatomic) NSNumber *alive;
@property (copy, nonatomic) NSNumber *musicOn;
@property (copy, nonatomic) NSNumber *sfxOn;
@property (copy, nonatomic) NSNumber *goreOn;
@property (copy, nonatomic) NSNumber *pausedVariable;
@property (copy, nonatomic) NSNumber *currentScore;
@property (copy, nonatomic) NSNumber *pausedMenu;
@property (copy, nonatomic) NSNumber *health;
@property (copy, nonatomic) NSNumber *ammo;
@property (copy, nonatomic) NSNumber *currentLevel;
@property (copy, nonatomic) NSNumber *saved;
@property (copy, nonatomic) NSString *errorMessage;

@property (copy, nonatomic) NSMutableArray *highScores;
@property (copy, nonatomic) NSMutableArray *highScoreNames;

@property (copy,nonatomic) NSNumber *playerX;
@property (copy,nonatomic) NSNumber *playerY;

@property (copy, nonatomic) NSNumber *startingNewGame;

+ (id) sharedInstance;

@end
```

```
//
//  Engine.m
//  ALevelProject
//
//  Created by Daniel Ellis.
//  Copyright 2014 Daniel Ellis. All rights reserved.
//

#import "Engine.h"

@implementation Engine

@synthesize alive;
@synthesize musicOn;
@synthesize sfxOn;
@synthesize goreOn;
@synthesize pausedVariable;
@synthesize currentScore;
@synthesize pausedMenu;
@synthesize health;
@synthesize ammo;
@synthesize currentLevel;
@synthesize saved;
@synthesize errorMessage;

@synthesize highScores;
@synthesize highScoreNames;

@synthesize playerX;
@synthesize playerY;

@synthesize startingNewGame;


- (id) init
{
    //initialise singleton
    if (self = [super init])
    {
        [self setPausedVariable:[NSNumber numberWithBool:false]];
        [self setPausedMenu:[NSNumber numberWithBool:false]];
        NSNumber *temp = [NSNumber numberWithInt:0];
        [self setHighScores:[NSMutableArray arrayWithObjects:temp,temp,temp,
            temp,temp, nil]];
        highScores = [highScores mutableCopy];
        NSString *temp2 = @"-";
        [self setHighScoreNames:[NSMutableArray arrayWithObjects:temp2,temp2,
            temp2,temp2,temp2, nil]];
        highScoreNames = [highScoreNames mutableCopy];
    }
    return self;
}

+ (id) sharedInstance
{
    //return singleton
    static dispatch_once_t p = 0;

    __strong static id _sharedObject = nil;
```

```
    dispatch_once(&p, ^{
        _sharedObject = [[self alloc] init];
    });

    return _sharedObject;
}


@end
```

```
//
//  ErrorLayer.h
//  ALevelProject
//
//  Created by Daniel Ellis.
//  Copyright 2014 Daniel Ellis. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "cocos2d.h"

@interface ErrorLayer : CCLayer {

}

+(CCScene *) scene;

@end
```

```
//
//  ErrorLayer.m
//  ALevelProject
//
//  Created by Daniel Ellis.
//  Copyright 2014 Daniel Ellis. All rights reserved.
//

#import "ErrorLayer.h"
#import "GameOverLayer.h"
#import "Engine.h"


@implementation ErrorLayer

+(CCScene *) scene
{
    //return the scene
    CCScene *scene = [CCScene node];
    ErrorLayer *layer = [ErrorLayer node];
    [scene addChild: layer];
    return scene;
}

-(id) init
{
    //initialise the error layer
    if( (self=[super init]) )
    {
        CCLabelTTF *error = [CCLabelTTF labelWithString:@"Error"
            fontName:@"Marker Felt" fontSize:64];

        CGSize size = [[CCDirector sharedDirector] winSize];
        error.position =  ccp( size.width /2 , 4*(size.height/5) );
        [self addChild: error];

        CCLabelTTF *errorText = [CCLabelTTF labelWithString:[[Engine
            sharedInstance] errorMessage] fontName:@"Marker Felt" fontSize:28];

        errorText.position =  ccp( size.width /2 , size.height/2 );
        [self addChild: errorText];

        [CCMenuItemFont setFontSize:28];

        CCMenuItem *itemBack = [CCMenuItemFont itemWithString:@"OK" target:
            self selector:@selector(back)];

        CCMenu *backMenu = [CCMenu menuWithItems:itemBack, nil];
        [backMenu setPosition:ccp( size.width/2, size.height/4)];
        [self addChild:backMenu];

        CCParticleRain* emitter = [[CCParticleRain alloc] init];
        emitter.texture = [[CCTextureCache sharedTextureCache]
            addImage:@"snow.png"];
        emitter.startColor = ccc4f(0, 0, 1, 0.8);
        emitter.startSize = 2;
        emitter.startSpin = 0;
        emitter.emissionRate = 100;
        emitter.radialAccel = 0;
        emitter.tangentialAccel = 0;
```

```
        emitter.zOrder = -1;
        emitter.speed = 200;

        emitter.position = ccp(size.width/2,size.height);

        [self addChild:emitter];
    }
    return self;
}

- (void) back
{
    //return to the game over layer
    [[CCDirector sharedDirector] replaceScene:[CCTransitionFade
        transitionWithDuration:1.0 scene:[GameOverLayer scene] withColor:
        ccBLACK]];
}

@end
```

```
//
//  GameLayer.h
//  ALevelProject
//
//  Created by Daniel Ellis.
//  Copyright 2014 Daniel Ellis. All rights reserved.
//

#import <Foundation/Foundation.h>
#import "cocos2d.h"
#import <GameKit/GameKit.h>

@interface GameLayer : CCLayer {


}

- (void) updatePlayer:(double) vX y:(double) vY d:(double) direction;
- (void) fireWithDamage:(int) damage direction:(int)direction accuracy:(int)
    accuracy;
- (int) checkForHits;
- (void) checkForBulletCollision;
- (void) updateEnemies;
- (void) showTrail:(double)direction trail:(bool)trailNumber;
- (void) trailRemover;
- (void) runAim;
- (void) runShoot;
- (void) runReload;
- (void) died;
- (void) save;
- (void) constructMap;

@end
```

```
//
//  GameLayer.m
//  ALevelProject
//
//  Created by Daniel Ellis.
//  Copyright 2014 Daniel Ellis. All rights reserved.
//

#import "GameLayer.h"
#import "Engine.h"
#import "AppDelegate.h"
#import "HUDLayer.h"
#import "Bullet.h"
#import "Enemy.h"
#import "HKTMXTiledMap.h"
#import "HKTMXLayer.h"
#import "BSPNode.h"

@interface GameLayer()

@property (strong) CCTMXTiledMap *tileMap;
@property (strong) CCTMXTiledMap *shadowMap;
@property (strong) CCTMXLayer *background;
@property (strong) CCTMXLayer *shadows;
@property (strong) CCSprite *player;
@property (strong) NSMutableSet *bullets;
@property (strong) NSMutableSet *enemys;
@property (strong) CCProgressTimer *trail1;
@property (strong) CCProgressTimer *trail2;
@property (strong) CCAction *aim;
@property (strong) CCAction *shot;
@property (strong) CCAction *deadAnimation;
@property (strong) CCAction *stopShoot;
@property (strong) CCAction *reload;
@property (strong) NSNumber *previousTilex;
@property (strong) NSNumber *previousTiley;
@property (strong) BSPNode *topNode;
@property (strong) NSNumber *justStarted;
@property (strong) Enemy *currentEnemy;

@end

@implementation GameLayer

- (id) init
{
    //initialise the game layer
    if( (self=[super init]) )
    {
        self.isTouchEnabled = YES;
        _trail1 = [CCProgressTimer progressWithSprite:[CCSprite
            spriteWithFile:@"trail.png"]];
        _trail1.type = kCCProgressTimerTypeBar;
        _trail1.percentage = 0;
        [self addChild:_trail1];
        _trail2 = [CCProgressTimer progressWithSprite:[CCSprite
            spriteWithFile:@"trail.png"]];
        _trail2.type = kCCProgressTimerTypeBar;
        _trail2.percentage = 0;
        [self addChild:_trail2];
```

```
        _justStarted = [NSNumber numberWithBool:YES];

        [self loadLevel];
        _justStarted = [NSNumber numberWithBool:NO];
    }
    return self;
}


- (void)setViewPointCenter:(CGPoint) position
{
    //show the tiles surrounding the player
    CGSize winSize = [CCDirector sharedDirector].winSize;

    int x = MAX(position.x, winSize.width/2);
    int y = MAX(position.y, winSize.height/2);
    x = MIN(x, (_tileMap.mapSize.width * _tileMap.tileSize.width) - winSize.
        width / 2);
    y = MIN(y, (_tileMap.mapSize.height * _tileMap.tileSize.height) - winSize.
        height/2);
    CGPoint actualPosition = ccp(x, y);

    CGPoint centerOfView = ccp(winSize.width/2, winSize.height/2);
    CGPoint viewPoint = ccpSub(centerOfView, actualPosition);
    self.position = viewPoint;
}

- (void) updatePlayer:(double) vX y:(double) vY d:(double) direction
{
    //update the player's position and rotation based on joystick input
    CGPoint playerPos = _player.position;

    int x = _player.position.x;
    int y = (_tileMap.mapSize.height * _tileMap.tileSize.height) - _player.
        position.y;

    //i collision points are with only the x component of the velocity
    //j collision points are with only the y component of the velocity
    int tempx =(x + vX + 8)/8;
    int tempy =(y + 7)/8;
    CGPoint iNEPoint = ccp(tempx,tempy);
    tempx =(x + vX - 8)/8;
    tempy =(y + 7)/8;
    CGPoint iNWPoint = ccp(tempx,tempy);
    tempx =(x + vX + 8)/8;
    tempy =(y - 7)/8;
    CGPoint iSEPoint = ccp(tempx,tempy);
    tempx =(x + vX - 8)/8;
    tempy =(y - 7)/8;
    CGPoint iSWPoint = ccp(tempx,tempy);
    tempx =(x + 7)/8;
    tempy =(y - vY - 8)/8;
    CGPoint jNEPoint = ccp(tempx,tempy);
    tempx =(x - 7)/8;
    tempy =(y - vY - 8)/8;
    CGPoint jNWPoint = ccp(tempx,tempy);
    tempx =(x + 7)/8;
    tempy =(y - vY + 8)/8;
    CGPoint jSEPoint = ccp(tempx,tempy);
    tempx =(x - 7)/8;
```

```
tempy =(y - vY + 8)/8;
CGPoint jSWPoint = ccp(tempx,tempy);

BOOL iNECollision = [[_tileMap propertiesForGID:[_background tileGIDAt:
    iNEPoint]][@"Collidable"]isEqualToString:@"True"];
BOOL iNWCollision = [[_tileMap propertiesForGID:[_background tileGIDAt:
    iNWPoint]][@"Collidable"]isEqualToString:@"True"];
BOOL iSECollision = [[_tileMap propertiesForGID:[_background tileGIDAt:
    iSEPoint]][@"Collidable"]isEqualToString:@"True"];
BOOL iSWCollision = [[_tileMap propertiesForGID:[_background tileGIDAt:
    iSWPoint]][@"Collidable"]isEqualToString:@"True"];
BOOL jNECollision = [[_tileMap propertiesForGID:[_background tileGIDAt:
    jNEPoint]][@"Collidable"]isEqualToString:@"True"];
BOOL jNWCollision = [[_tileMap propertiesForGID:[_background tileGIDAt:
    jNWPoint]][@"Collidable"]isEqualToString:@"True"];
BOOL jSECollision = [[_tileMap propertiesForGID:[_background tileGIDAt:
    jSEPoint]][@"Collidable"]isEqualToString:@"True"];
BOOL jSWCollision = [[_tileMap propertiesForGID:[_background tileGIDAt:
    jSWPoint]][@"Collidable"]isEqualToString:@"True"];

if (jNECollision || jNWCollision)
{
    tempy = (playerPos.y + vY + 1)/8;
    playerPos.y = (tempy * 8);
}
else if (jSECollision || jSWCollision)
{
    tempy = (playerPos.y + vY - 1)/8;
    playerPos.y = (tempy * 8) + 8;
}
else
{
    playerPos.y += vY;
}

if (iNECollision || iSECollision)
{
    tempx = (playerPos.x + vX + 1)/8;
    playerPos.x = (tempx * 8);
}
else if (iNWCollision || iSWCollision)
{
    tempx = (playerPos.x + vX -1)/8;
    playerPos.x = (tempx * 8) + 8;
}
else
{
    playerPos.x += vX;
}

_player.position = playerPos;
_player.rotation = direction;
[self setViewPointCenter:playerPos];

//crossing to a different tile
if (_previousTilex.integerValue != (int)playerPos.x/8 || _previousTiley.
    integerValue != (int)playerPos.y/8)
{
    [self performSelectorInBackground:@selector(FOV) withObject:nil];
    _previousTilex = [NSNumber numberWithInt:playerPos.x/8];
```

```
        _previousTiley = [NSNumber numberWithInt:playerPos.y/8];
        int GID = _background.tileset.firstGid;

        if ([_background tileGIDAt:ccp((int)playerPos.x/8, _tileMap.mapSize.
            height - ((int)playerPos.y/8))] >= GID+37 && [_background
            tileGIDAt:ccp((int)playerPos.x/8, _tileMap.mapSize.height - ((int)
            playerPos.y/8))] <= GID+40)
        {
            // player has stepped on the stairs to the next level
            [[Engine sharedInstance] setCurrentScore:[NSNumber numberWithInt:
                [[Engine sharedInstance] currentScore].intValue + 10]];
            [[Engine sharedInstance] setCurrentLevel:[NSNumber numberWithInt:
                [[Engine sharedInstance] currentLevel].intValue + 1]];
            [[Engine sharedInstance] setHealth:[NSNumber numberWithInt:100]];

            for (Enemy *enemy in _enemys.allObjects)
            {
                if (enemy != nil)
                {
                    [self removeChild:enemy cleanup:YES];
                    [self removeChild:enemy.healthDisplayReference cleanup:YES
                        ];
                }
            }
            for (int x = 150; x <=350; x++)
            {
                for (int y = 150; y <=350; y++)
                {
                    [_background setTileGID:(_background.tileset.firstGid) at:
                        ccp(x,y)];
                    [_shadows setTileGID:(_shadows.tileset.firstGid) at:ccp(x,
                        y)];
                }
            }
            _enemys = nil;
            [self constructMap];
            [_shadows setTileGID:_shadows.tileset.firstGid + 1 at:ccp(_player.
                position.x/8,_tileMap.mapSize.height - (_player.position.y/8))]
                ;
            [self FOV];
            return;
        }
    }
    [[Engine sharedInstance] setPlayerX:[NSNumber numberWithInt:_player.
        position.x]];
    [[Engine sharedInstance] setPlayerY:[NSNumber numberWithInt:_player.
        position.y]];
}

- (void) fireWithDamage:(int)damage direction:(int)direction accuracy:(int)
    accuracy
{
    //fire a bullet
    Bullet *bullet = [Bullet spriteWithFile:@"Bullet.png"];
    bullet.position = _player.position;

    float i = arc4random()%200;
    i = ((i + 1)/100)-1;
    float angle = ((100-accuracy)*40*i)/100;
```

```
    [bullet setupWithDamage:damage direction:direction + angle];
    NSMutableArray *objectsArray;

    if (_bullets != nil)
    {
        objectsArray = [NSMutableArray arrayWithArray: _bullets.allObjects];
        [objectsArray addObject:bullet];
        _bullets = [NSMutableSet setWithArray:objectsArray];
    }
    else
    {
        _bullets = [NSMutableSet setWithObject:bullet];
    }

    [self addChild: bullet];

    CCParticleExplosion* emitter2 = [[CCParticleExplosion alloc] init];
    emitter2.texture = [[CCTextureCache sharedTextureCache]
        addImage:@"snow.png"];
    emitter2.startSize = 1;
    emitter2.endSize = 2;
    emitter2.duration = 0.1;
    emitter2.life = 0.1;
    emitter2.lifeVar = 0.1;
    emitter2.speed = 100;
    emitter2.speedVar = 10;
    emitter2.angle = 90-bullet.direction.intValue;
    emitter2.angleVar = 10;
    emitter2.startColor = ccc4f(1, 0.3, 0, 0.1);
    emitter2.startColorVar = ccc4f(0, 0.3, 0, 0);
    emitter2.endColor = ccc4f(1, 0.3, 0, 0.1);
    emitter2.endColorVar = ccc4f(0, 0.3, 0, 0);
    emitter2.position = ccp(_player.position.x,_player.position.y);
    emitter2.zOrder = 3;
    emitter2.emissionRate = 1000;
    [self addChild:emitter2];
}

- (int) checkForHits
{
    // check whether the enemys have attacked the player
    int damage = 0;
    for (Enemy *enemy in _enemys.allObjects)
    {
        if (enemy != nil)
        {
            if ([enemy checkForAttack])
            {
                damage += 5;
            }
        }
    }
    return damage;
}

- (void) checkForBulletCollision
{
    //check if any of the bullets have hit any of the enemies or walls
    if(_bullets)
    {
```

```
CGPoint coord;
int x;
int y;
int GID;
NSDictionary *properties;
NSString *collision;

for (Bullet *bullet in _bullets.allObjects)
{
    if (bullet != nil)
    {
        if (bullet.dead.boolValue == false)
        {
            coord = [bullet futurePos];
            x = coord.x / 8;
            y = ((_tileMap.mapSize.height * _tileMap.tileSize.height)
                - coord.y) / 8;
            GID = [_background tileGIDAt:ccp(x,y)];
            properties = [_tileMap propertiesForGID:GID];
            collision = properties[@"Collidable"];

            if ([collision isEqualToString:@"True"])
            {
                //collided with wall
                [bullet stopped];
                [self removeChild:bullet cleanup:YES];
            }
            else
            {
                [bullet updatePosition:coord];
                for (Enemy *enemy __strong in _enemys.allObjects)
                {
                    if (enemy.dead.boolValue == false && bullet.
                        position.x <= enemy.position.x + 15 && bullet.
                        position.x >= enemy.position.x - 15 && bullet.
                        position.y <= enemy.position.y + 15 && bullet.
                        position.y >= enemy.position.y - 15)
                    {
                        //collided with enemy
                        bool dead = [enemy hitWithDamageDead:bullet.
                            damage.integerValue Direction:bullet.
                            direction.integerValue Map:_tileMap Layer:
                            _background];

                        [bullet stopped];
                        [enemy healthDisplayReference].percentage =
                            (100*enemy.health.integerValue) / (100 +
                            ([[Engine sharedInstance] currentLevel].
                            intValue*100));

                        if ([[Engine sharedInstance] goreOn].boolValue
                            == YES)
                        {
                            CCParticleExplosion* emitter2 =
                                [[CCParticleExplosion alloc] init];
                            emitter2.texture = [[CCTextureCache
                                sharedTextureCache]
                                addImage:@"snow.png"];
                            emitter2.startSize = 1;
                            emitter2.duration = 0.1;
```

```
                                    emitter2.life = 0.08;
                                    emitter2.lifeVar = 0.1;
                                    emitter2.speed = 100;
                                    emitter2.speedVar = 10;
                                    emitter2.angle = 90-bullet.direction.
                                        intValue;
                                    emitter2.angleVar = 15;
                                    emitter2.startColor = ccc4f(1, 0, 0, 0.1);
                                    emitter2.startColorVar = ccc4f(0.5, 0, 0,
                                        0);
                                    emitter2.endColor = ccc4f(1, 0, 0, 0.1);
                                    emitter2.endColorVar = ccc4f(0.5, 0, 0, 0)
                                        ;
                                    emitter2.position = ccp(enemy.position.x,
                                        enemy.position.y);
                                    emitter2.zOrder = 3;
                                    emitter2.emissionRate = 1000;

                                    [self addChild:emitter2];
                                }

                                if (dead)
                                {
                                    [[Engine sharedInstance] setCurrentScore:
                                        [NSNumber numberWithInt:[[Engine
                                        sharedInstance] currentScore].intValue
                                        + 1]];
                                    _currentEnemy = enemy;

                                    [enemy runAction:_deadAnimation];
                                }

                                [self removeChild:bullet cleanup:YES];
                                break;
                            }
                        }
                    }
                }
            }
        }
    }
}

- (void) changeEnemy
{
    //change the enemy's sprite to indicate that it is dead
    [_currentEnemy setTexture:[[CCTextureCache sharedTextureCache]
        addImage:@"enemydead.png"]];
}

- (void) loadLevel
{
    //set up the level and load it from memory if necessary
    _tileMap = [HKTMXTiledMap tiledMapWithTMXFile:@"mainMap.tmx"];
    _background = [_tileMap layerNamed:@"Background"];

    [[CCSpriteFrameCache sharedSpriteFrameCache]
        addSpriteFramesWithFile:@"playersheet.plist"];
    CCSpriteBatchNode *playerSheet = [CCSpriteBatchNode
        batchNodeWithFile:@"playersheet.png"];
```

```
    [self addChild:playerSheet];

    _player =  [CCSprite spriteWithSpriteFrameName:@"playernatural.png"];

    [self addChild:_tileMap z:(-1)];
    _shadowMap = [HKTMXTiledMap tiledMapWithTMXFile:@"shadowMapLowRes.tmx"];
    _shadows = [_shadowMap layerNamed:@"Shadows"];
    [self addChild:_shadowMap z:2];
    if (![[Engine sharedInstance] alive].boolValue)
    {
        //create new map
        [self constructMap];
        [[Engine sharedInstance] setAmmo:[NSNumber numberWithInt:10]];
    }
    else
    {
        _player.position = ccp([[Engine sharedInstance] playerX].intValue,
            [[Engine sharedInstance] playerY].intValue);

        if (_justStarted.boolValue)
        {
            //load map from memory
            [[Engine sharedInstance] setAmmo:[NSNumber numberWithInt:10]];
            NSArray *paths = NSSearchPathForDirectoriesInDomains
                (NSDocumentDirectory, NSUserDomainMask, YES);
            NSString *documentsDirectoryPath = [paths objectAtIndex:0];
            NSString *filePath = [documentsDirectoryPath
                stringByAppendingPathComponent:@"appData"];

            if ([[NSFileManager defaultManager] fileExistsAtPath:filePath])
            {
                NSData *data = [NSData dataWithContentsOfFile:filePath];
                NSDictionary *savedData = [NSKeyedUnarchiver
                    unarchiveObjectWithData:data];
                NSArray *array = [savedData objectForKey:@"tiles"];

                for (int x = 200; x < 300; x++)
                {
                    for (int y = 200; y < 300; y++)
                    {
                        NSNumber *number = [array objectAtIndex:((x-200)*100)+
                            (y-200)];
                        [_background setTileGID:number.intValue at:ccp(x,y)];
                    }
                }

                array = [savedData objectForKey:@"enemies"];
                for (int x = 0; x < [array count]/5; x++)
                {
                    NSNumber *tempObject = [array objectAtIndex:x*5];
                    NSNumber *tempObject2 = [array objectAtIndex:(x*5)+1];
                    NSNumber *tempObject3 = [array objectAtIndex:(x*5)+2];
                    Enemy *enemy;
                    if (tempObject3.intValue <= 0)
                    {
                        enemy = [Enemy
                            spriteWithSpriteFrameName:@"enemydead.png"];
                    }
                    else
                    {
```

```
                    enemy = [Enemy
                        spriteWithSpriteFrameName:@"enemynatural.png"];
                }

                enemy.position = ccp(tempObject.intValue,tempObject2.
                    intValue);

                tempObject = [array objectAtIndex:(x*5)+2];
                [enemy initWithHealth:tempObject3.intValue Damage:1];

                if (enemy.health.intValue <= 0)
                {
                    enemy.dead = [NSNumber numberWithBool:YES];
                }

                tempObject = [array objectAtIndex:(x*5)+3];
                enemy.activated = [NSNumber numberWithBool:tempObject.
                    boolValue];

                tempObject = [array objectAtIndex:(x*5)+4];
                enemy.rotation = tempObject.floatValue;

                [self addChild:enemy z:1];
                [self addChild:[enemy healthDisplayReference] z:1];

                NSMutableArray *objectsArray;

                if (_enemys != nil)
                {
                    objectsArray = [NSMutableArray arrayWithArray: _enemys
                        .allObjects];
                    [objectsArray addObject:enemy];
                    _enemys = [NSMutableSet setWithArray:objectsArray];
                }
                else
                {
                    _enemys = [NSMutableSet setWithObject:enemy];
                }
            }
            NSError *error;
            [[NSFileManager defaultManager] removeItemAtPath:filePath
                error:&error];
        }
    }
}

//setup animations
NSMutableArray *aimFrames = [NSMutableArray array];
for (int i = 1; i <= 7; i++)
{
    [aimFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache]
        spriteFrameByName:[NSString stringWithFormat:@"playeraim%d.png",i]]
        ];
}

[playerSheet addChild: _player];
_previousTilex = [NSNumber numberWithInt:_player.position.x/8];
_previousTiley = [NSNumber numberWithInt:_player.position.y/8];

CCAnimation *aim = [CCAnimation animationWithSpriteFrames:aimFrames delay:
```

```
        0.08];
    aim.loops = 1;
    _aim = [CCAnimate actionWithAnimation:aim];


    NSMutableArray *shotFrames = [NSMutableArray array];
    for (int i = 7; i >= 1; i--)
    {
        [shotFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache]
            spriteFrameByName:[NSString stringWithFormat:@"playeraim%d.png",i]]
            ];
    }
    [shotFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache]
        spriteFrameByName:@"playernatural.png"]];
    CCAnimation *shot = [CCAnimation animationWithSpriteFrames:shotFrames
        delay:0.08];
    shot.loops = 1;
    _shot = [CCAnimate actionWithAnimation:shot];

    NSMutableArray *deadAnimationFrames = [NSMutableArray array];
    [deadAnimationFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache
        ] spriteFrameByName:@"enemydead.png"]];
    CCAnimation *deadAnimation = [CCAnimation animationWithSpriteFrames:
        deadAnimationFrames delay:0.08];
    _deadAnimation = [CCAnimate actionWithAnimation:deadAnimation];

    NSMutableArray *reloadFrames = [NSMutableArray array];
    for (int i = 1; i <= 7; i++)
    {
        [reloadFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache]
            spriteFrameByName:[NSString stringWithFormat:@"playeraim%d.png",i]]
            ];
    }
    for (int i = 1; i <= 7; i++)
    {
        [reloadFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache]
            spriteFrameByName:[NSString stringWithFormat:@"playerreload%d.png",
            i]]];
    }
    for (int i = 7; i >= 1; i--)
    {
        [reloadFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache]
            spriteFrameByName:[NSString stringWithFormat:@"playeraim%d.png",i]]
            ];
    }
    [reloadFrames addObject:[[CCSpriteFrameCache sharedSpriteFrameCache]
        spriteFrameByName:@"playernatural.png"]];
    CCAnimation *reload = [CCAnimation animationWithSpriteFrames:reloadFrames
        delay:0.08];
    reload.loops = 1;
    _reload = [CCAnimate actionWithAnimation:reload];

    //light the starting tile
    int GID = _shadows.tileset.firstGid + 1;
    [_shadows setTileGID:GID at:ccp(_player.position.x/8,_tileMap.mapSize.
        height - (_player.position.y/8))];
    [self FOV];
}

- (void) updateEnemies
```

```
{
    //update the positions of the enemies and their health bars
    for (Enemy *enemy in _enemys.allObjects)
    {
        if (enemy != nil)
        {
            [enemy movement:_player.position.x :_player.position.y Layer:
                _background Map:_tileMap];
            [enemy healthDisplayReference].position = ccp(enemy.position.x,
                enemy.position.y+10);
        }
    }
}

- (void) showTrail:(double)direction trail:(bool)trailNumber
{
    //display the two aiming indicator lines
    double fracx = sin(direction*((2*3.141592654)/360));
    double fracy = cos(direction*((2*3.141592654)/360));
    int x;
    int y;
    int GID;
    NSDictionary *properties;
    NSString *collision;
    int range = 0;
    for (int n = 1; n <= 590; n++)
    {
        x = (_player.position.x + (n*fracx))/8;
        y = ((_tileMap.mapSize.height * _tileMap.tileSize.height) - _player.
            position.y - (n*fracy))/8;
        GID = [_background tileGIDAt:ccp(x,y)];
        properties = [_tileMap propertiesForGID:GID];
        collision = properties[@"Collidable"];

        if ([collision isEqualToString:@"True"])
        {
            range = n - 1;
            break;
        }
    }
    if (trailNumber)
    {
        [self removeChild:_trail1 cleanup:YES];
        _trail1.midpoint = ccp(0,0);
        _trail1.barChangeRate = ccp(0,1);
        _trail1.anchorPoint = ccp(0,0);
        _trail1.percentage = range/3;
        _trail1.rotation = direction;
        _trail1.position = ccp(_player.position.x, _player.position.y);
        [self addChild:_trail1];
    }
    else
    {
        [self removeChild:_trail2 cleanup:YES];
        _trail2.midpoint = ccp(0,0);
        _trail2.barChangeRate = ccp(0,1);
        _trail2.anchorPoint = ccp(0,0);
        _trail2.percentage = range/3;
        _trail2.rotation = direction;
        _trail2.position = ccp(_player.position.x, _player.position.y);
```

```
        [self addChild:_trail2];
    }
}

-(void) trailRemover
{
    //remove the aiming indicator lines
    _trail1.percentage = 0;
    _trail2.percentage = 0;
}

- (void) runAim
{
    //run the aiming animation of the player
    [_player runAction:_aim];
}

- (void) runShoot
{
    //run the shooting animation of the player
    [_player runAction:_shot];
}

- (void) runReload
{
    //run the reload animation of the player
    [_player runAction:_reload];
}

- (void) runStopShoot
{
    //run the animation of the player holstering their gun
    [_player runAction:_stopShoot];
}

- (void) FOV
{
    // execute the shadow algorithm in each octant
    [self castLightRow:1 StartGradient:0 EndGradient:1 xx:1 xy:0 yx:0 yy:1];//
        ENE octant
    [self castLightRow:1 StartGradient:0 EndGradient:1 xx:0 xy:1 yx:1 yy:0];//
        NNE octant
    [self castLightRow:1 StartGradient:0 EndGradient:1 xx:0 xy:-1 yx:1 yy:
        0];//NNW octant
    [self castLightRow:1 StartGradient:0 EndGradient:1 xx:-1 xy:0 yx:0 yy:
        1];//WNW octant
    [self castLightRow:1 StartGradient:0 EndGradient:1 xx:-1 xy:0 yx:0
        yy:-1];//WSW octant
    [self castLightRow:1 StartGradient:0 EndGradient:1 xx:0 xy:-1 yx:-1 yy:
        0];//SSW octant
    [self castLightRow:1 StartGradient:0 EndGradient:1 xx:0 xy:1 yx:-1 yy:
        0];//SSE octant
    [self castLightRow:1 StartGradient:0 EndGradient:1 xx:1 xy:0 yx:0
        yy:-1];//ESE octant
}

- (void) castLightRow:(int)row StartGradient:(float)startGradient EndGradient:
    (float)endGradient xx:(int)xx xy:(int)xy yx:(int)yx yy:(int)yy
{
    //execute the shadow algorithm for the spcified octant
```

```
bool previousCollidable = false;
float newStartGradient = 0;
for (float x = row; x <= 20 && !previousCollidable; x++)
{
    for (float y = 0; y <= x; y++)
    {
        float topGradient = (y+0.5)/(x-0.5); //gradient to top left of the
            tile
        float bottomGradient = (y-0.5)/(x+0.5); //gradient to bottom right
            of the tile
        if (topGradient >= startGradient)
        {
            //not in a shadow
            if (endGradient < bottomGradient)
            {
                //reached the diagonal or the bottom of a shadow
                break;
            }

            //transformation to the map coordinates for the correct octant
            int tempX = _player.position.x/8 + (x * xx) + (y * xy);
            int tempY = _player.position.y/8 + (x * yx) + (y * yy);
            tempY = _tileMap.mapSize.height - tempY;

            if ((x*x) + (y*y) <= 400)
            {
                //tile is within a radius of 20 tiles, set tile as visible
                int GID = _shadows.tileset.firstGid;
                [_shadows setTileGID:GID+1 at:ccp(tempX,tempY)];
            }

            if (previousCollidable)
            {
                if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp
                    (tempX,tempY)]][@"Collidable"]isEqualToString:@"True"])
                {
                    newStartGradient = topGradient;
                }
                else
                {
                    previousCollidable = false;
                    startGradient = newStartGradient;
                }
            }
            else
            {
                if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp
                    (tempX,tempY)]][@"Collidable"]isEqualToString:@"True"]
                    && x < 20)
                {
                    previousCollidable = true;
                    if (startGradient <= bottomGradient)
                    {
                        [self castLightRow:x + 1 StartGradient:
                            startGradient EndGradient:bottomGradient xx:xx
                            xy:xy yx:yx yy:yy];
                    }
                    newStartGradient = topGradient;
                }
            }
```

```
            }
        }
    }
}

-(void) constructMap
{
    // generate binary space partition tree
    _topNode = [BSPNode new];
    _topNode.topRightx = [NSNumber numberWithInt:300];
    _topNode.topRighty = [NSNumber numberWithInt:300];
    _topNode.botLeftx = [NSNumber numberWithInt:200];
    _topNode.botLefty = [NSNumber numberWithInt:200];
    int n = 8;
    [self split:_topNode n:n];

    //populate rooms with enemies and spawn the player
    [self setSpawn:_topNode];
    [self spawnEnemies:_topNode];

    //link up rooms
    [self linkRooms:_topNode];

    //add walls to the rooms
    int GID = _background.tileset.firstGid;
    int adjacents;
    BOOL top;
    BOOL bot;
    BOOL right;
    BOOL left;
    for (int x = 200; x< 300; x++)
    {
        for (int y = 200; y<300; y++)
        {
            if([[_tileMap propertiesForGID:[_background tileGIDAt:ccp(x,
                _tileMap.mapSize.height - y)]]
                [@"Collidable"]isEqualToString:@"True"])
            {
                top = 0;
                bot = 0;
                right = 0;
                left = 0;
                adjacents = 0;
                if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp(x+1
                    ,_tileMap.mapSize.height - y)]][@"Collidable"]
                    isEqualToString:@"False"])
                {
                    adjacents ++;
                    right = 1;
                }
                if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp(x-1
                    ,_tileMap.mapSize.height- y)]][@"Collidable"]
                    isEqualToString:@"False"])
                {
                    adjacents ++;
                    left = 1;
                }
                if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp(x,
                    _tileMap.mapSize.height - y - 1)]][@"Collidable"]
                    isEqualToString:@"False"])
```

```
{
    adjacents ++;
    top = 1;
}
if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp(x,
    _tileMap.mapSize.height - y + 1)]][@"Collidable"]
    isEqualToString:@"False"])
{
    adjacents ++;
    bot = 1;
}

if (adjacents != 0)
{
    if (adjacents == 1)
    {
        //one adjacent walkable tile
        if (right)
        {
            [_background setTileGID:GID+24 at:ccp(x,_tileMap.
                mapSize.height -y)];
        }
        if (left)
        {
            [_background setTileGID:GID+23 at:ccp(x,_tileMap.
                mapSize.height -y)];
        }
        if (top)
        {
            [_background setTileGID:GID+28 at:ccp(x,_tileMap.
                mapSize.height -y)];
        }
        if (bot)
        {
            [_background setTileGID:GID+29 at:ccp(x,_tileMap.
                mapSize.height -y)];
        }
    }
    else if (adjacents == 3)
    {
        //3 adjacent walkable tiles
        if (y%2)
        {
            [_background setTileGID:GID+14 at:ccp(x,_tileMap.
                mapSize.height - y)];
        }
        else
        {
            [_background setTileGID:GID+16 at:ccp(x,_tileMap.
                mapSize.height - y)];
        }
    }
    else if (adjacents == 2)
    {
        //2 adjacent walkable tiles
        if (right)
        {
            if (left)
            {
                if (y%2)
```

```
                    {
                        [_background setTileGID:GID+14 at:ccp(x,
                            _tileMap.mapSize.height - y)];
                    }
                    else
                    {
                        [_background setTileGID:GID+16 at:ccp(x,
                            _tileMap.mapSize.height - y)];
                    }
                }
                if (top)
                {
                    [_background setTileGID:GID+25 at:ccp(x,
                        _tileMap.mapSize.height -y)];
                }
                if (bot)
                {
                    [_background setTileGID:GID+34 at:ccp(x,
                        _tileMap.mapSize.height -y)];
                }
            }
            else if (left)
            {
                if (top)
                {
                    [_background setTileGID:GID+36 at:ccp(x,
                        _tileMap.mapSize.height -y)];
                }
                if (bot)
                {
                    [_background setTileGID:GID+35 at:ccp(x,
                        _tileMap.mapSize.height -y)];
                }
            }
            else if (top)
            {
                if (bot)
                {
                    if (y%2)
                    {
                        [_background setTileGID:GID+14 at:ccp(x,
                            _tileMap.mapSize.height - y)];
                    }
                    else
                    {
                        [_background setTileGID:GID+16 at:ccp(x,
                            _tileMap.mapSize.height - y)];
                    }
                }
            }
        }
    }
    else if (adjacents == 4)
    {
        //4 adjacent walkable tiles
        if (y%2)
         {
             [_background setTileGID:GID+14 at:ccp(x,_tileMap.
                 mapSize.height - y)];
         }
```

```
                    else
                    {
                        [_background setTileGID:GID+16 at:ccp(x,_tileMap.
                            mapSize.height - y)];
                    }
                }
                else
                {
                    // diagonally adjacent walkable tiles
                    if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp
                        (x-1,_tileMap.mapSize.height- y + 1)]][@"Collidable"]
                        isEqualToString:@"False"])
                    {
                        [_background setTileGID:GID+42 at:ccp(x,_tileMap.
                            mapSize.height -y)];
                    }
                    else if ([[_tileMap propertiesForGID:[_background
                        tileGIDAt:ccp(x+1,_tileMap.mapSize.height- y - 1)]]
                        [@"Collidable"]isEqualToString:@"False"])
                    {
                        [_background setTileGID:GID+41 at:ccp(x,_tileMap.
                            mapSize.height -y)];
                    }
                    else if ([[_tileMap propertiesForGID:[_background
                        tileGIDAt:ccp(x-1,_tileMap.mapSize.height- y - 1)]]
                        [@"Collidable"]isEqualToString:@"False"])
                    {
                        [_background setTileGID:GID+27 at:ccp(x,_tileMap.
                            mapSize.height -y)];
                    }
                    else if ([[_tileMap propertiesForGID:[_background
                        tileGIDAt:ccp(x+1,_tileMap.mapSize.height- y + 1)]]
                        [@"Collidable"]isEqualToString:@"False"])
                    {
                        [_background setTileGID:GID+43 at:ccp(x,_tileMap.
                            mapSize.height -y)];
                    }
                    else
                    {
                        [_background setTileGID:GID at:ccp(x,_tileMap.mapSize.
                            height -y)];
                    }
                }
            }
        }
    }
    [self setExit:_topNode];
}

-(void) split:(BSPNode*)Node n:(int)n
{
    // split the inputted node into two nodes
    if (n>0)
    {
        BOOL vertical;
        if (Node.topRightx.intValue - Node.botLeftx.intValue < 25 || Node.
            topRighty.intValue - Node.botLefty.intValue < 25)
        {
            //create room because node is too small
            float fraction = arc4random()%26;
```

```
        fraction = (fraction/100)+0.75;
        float newVal = ((Node.topRightx.intValue - Node.botLeftx.intValue)
            *fraction)+Node.botLeftx.intValue - 1;
        Node.topRightx = [NSNumber numberWithInt:newVal];

        fraction = arc4random()%26;
        fraction = (fraction/100)+0.75;
        newVal = ((Node.topRighty.intValue - Node.botLefty.intValue)*
            fraction)+Node.botLefty.intValue - 1;
        Node.topRighty = [NSNumber numberWithInt:newVal];

        fraction = arc4random()%26;
        fraction = fraction/100;
        newVal = ((Node.topRightx.intValue - Node.botLeftx.intValue)*
            fraction)+Node.botLeftx.intValue + 1;
        Node.botLeftx = [NSNumber numberWithInt:newVal];

        fraction = arc4random()%26;
        fraction = fraction/100;
        newVal = ((Node.topRighty.intValue - Node.botLefty.intValue)*
            fraction)+Node.botLefty.intValue + 1;
        Node.botLefty = [NSNumber numberWithInt:newVal];

        int GID = _background.tileset.firstGid;
        for (int x = Node.botLeftx.intValue; x<Node.topRightx.intValue; x+
            +)
        {
            for (int y = Node.botLefty.intValue; y<Node.topRighty.intValue
                ;y++)
            {
                if (y%2)
                {
                    [_background setTileGID:GID+14 at:ccp(x,_tileMap.
                        mapSize.height - y)];
                }
                else
                {
                    [_background setTileGID:GID+16 at:ccp(x,_tileMap.
                        mapSize.height - y)];
                }
            }
        }
        return;
    }

    //create child nodes
    Node.leftNode = [BSPNode new];
    Node.rightNode = [BSPNode new];

    Node.leftNode.botLeftx = [NSNumber numberWithInt:Node.botLeftx.
        intValue];
    Node.leftNode.botLefty = [NSNumber numberWithInt:Node.botLefty.
        intValue];
    Node.rightNode.topRightx = [NSNumber numberWithInt:Node.topRightx.
        intValue];
    Node.rightNode.topRighty = [NSNumber numberWithInt:Node.topRighty.
        intValue];

    if (Node.topRightx.intValue - Node.botLeftx.intValue > 2*(Node.
        topRighty.intValue - Node.botLefty.intValue))
```

```
    {
        vertical = YES;
    }
    else if (Node.topRighty.intValue - Node.botLefty.intValue > 2*(Node.
        topRightx.intValue - Node.botLeftx.intValue))
    {
        vertical = NO;
    }
    else
    {
        vertical = arc4random()%2;
    }

    float division = arc4random()%61;
    division = (division/100)+0.2;

    //set coordinates of child nodes
    if (vertical)
    {
        Node.vertical = [NSNumber numberWithBool:YES];
        float newX = ((Node.topRightx.intValue - Node.botLeftx.intValue)*
            division)+Node.botLeftx.intValue;

        Node.split = [NSNumber numberWithInt:newX];

        Node.leftNode.topRightx = [NSNumber numberWithInt:newX];
        Node.rightNode.botLeftx = [NSNumber numberWithInt:Node.leftNode.
            topRightx.intValue];

        Node.leftNode.topRighty = [NSNumber numberWithInt:Node.topRighty.
            intValue];
        Node.rightNode.botLefty = [NSNumber numberWithInt:Node.botLefty.
            intValue];

    }
    else
    {
        Node.vertical = [NSNumber numberWithBool:NO];
        float newY = ((Node.topRighty.intValue - Node.botLefty.intValue)*
            division)+Node.botLefty.intValue;

        Node.split = [NSNumber numberWithInt:newY];

        Node.leftNode.topRighty =[NSNumber numberWithInt:newY];
        Node.rightNode.botLefty = [NSNumber numberWithInt:Node.leftNode.
            topRighty.intValue];

        Node.leftNode.topRightx = [NSNumber numberWithInt:Node.topRightx.
            intValue];
        Node.rightNode.botLeftx = [NSNumber numberWithInt:Node.botLeftx.
            intValue];
    }

    n--;
    [self split:Node.leftNode n:n];
    [self split:Node.rightNode n:n];
}
else
{
    //create rooms
```

```
            float fraction = arc4random()%26;
            fraction = (fraction/100)+0.75;
            float newVal = ((Node.topRightx.intValue - Node.botLeftx.intValue)*
                fraction)+Node.botLeftx.intValue - 1;
            Node.topRightx = [NSNumber numberWithInt:newVal];

            fraction = arc4random()%26;
            fraction = (fraction/100)+0.75;
            newVal = ((Node.topRighty.intValue - Node.botLefty.intValue)*fraction)
                +Node.botLefty.intValue - 1;
            Node.topRighty = [NSNumber numberWithInt:newVal];

            fraction = arc4random()%26;
            fraction = fraction/100;
            newVal = ((Node.topRightx.intValue - Node.botLeftx.intValue)*fraction)
                +Node.botLeftx.intValue + 1;
            Node.botLeftx = [NSNumber numberWithInt:newVal];

            fraction = arc4random()%26;
            fraction = fraction/100;
            newVal = ((Node.topRighty.intValue - Node.botLefty.intValue)*fraction)
                +Node.botLefty.intValue + 1;
            Node.botLefty = [NSNumber numberWithInt:newVal];

            int GID = _background.tileset.firstGid;
            for (int x = Node.botLeftx.intValue; x<Node.topRightx.intValue; x++)
            {
                for (int y = Node.botLefty.intValue; y<Node.topRighty.intValue;y++
                    )
                {
                    if (y%2)
                    {
                        [_background setTileGID:GID+14 at:ccp(x,_tileMap.mapSize.
                            height - y)];
                    }
                    else
                    {
                        [_background setTileGID:GID+16 at:ccp(x,_tileMap.mapSize.
                            height - y)];
                    }
                }
            }
        }
    }

    - (void) setSpawn:(BSPNode*)Node
    {
        //set the player's spawn
        if (Node.leftNode)
        {
            BOOL left = arc4random()%2;
            if (left)
            {
                [self setSpawn:Node.leftNode];
            }
            else
            {
                [self setSpawn:Node.rightNode];
            }
        }
```

```
    else
    {
        //set player spawn here
        _player.position = ccp(((Node.topRightx.intValue + Node.botLeftx.
            intValue)/2)*8,((Node.topRighty.intValue + Node.botLefty.intValue)/
            2)*8);
    }
}

- (void) linkRooms:(BSPNode*)Node
{
    //link the rooms together with corridors
    if (Node.leftNode.leftNode)
    {
        [self linkRooms:Node.leftNode];
    }
    if (Node.rightNode.leftNode)
    {
        [self linkRooms:Node.rightNode];
    }

    int GID = _background.tileset.firstGid;
    if (Node.vertical.boolValue)
    {
        //vertical split
        int topy = MIN([self maxTopRighty:Node.leftNode]-1, [self
            maxTopRighty:Node.rightNode]-1);
        int boty = MAX([self minBotLefty:Node.leftNode]+1,[self minBotLefty:
            Node.rightNode]+1);
        float fraction = arc4random()%91;
        fraction = (fraction/100)+0.05;
        fraction = (fraction*(topy - boty))+boty;
        int y = fraction;
        int GID2;
        int GID3;
        if (y%2)
        {
            GID2 = GID + 14;
            GID3 = GID + 16;
        }
        else
        {
            GID2 = GID + 16;
            GID3 = GID + 14;
        }
        int x = Node.split.intValue;
        //propogate path to the right
        for (;;x++)
        {
            if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp(x,
                _tileMap.mapSize.height - y)]]
                [@"Collidable"]isEqualToString:@"False"])
            {
                for (int tempX = x;tempX<x+3;tempX++)
                {
                    [_background setTileGID:GID2 at:ccp(tempX,_tileMap.mapSize
                        .height - y)];
                    [_background setTileGID:GID3 at:ccp(tempX,_tileMap.mapSize
                        .height - y - 1)];
                    [_background setTileGID:GID3 at:ccp(tempX,_tileMap.mapSize
```

```
                    .height - y + 1)];
                if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp
                    (tempX,_tileMap.mapSize.height - y - 3)]][@"Collidable"
                    ]isEqualToString:@"False"])
                {
                    [_background setTileGID:GID2 at:ccp(x,_tileMap.mapSize
                        .height - y - 2)];
                }
                if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp
                    (tempX,_tileMap.mapSize.height - y + 3)]][@"Collidable"
                    ]isEqualToString:@"False"])
                {
                    [_background setTileGID:GID2 at:ccp(x,_tileMap.mapSize
                        .height - y + 2)];
                }
            }
            break;
        }

        [_background setTileGID:GID2 at:ccp(x,_tileMap.mapSize.height - y)
            ];
        [_background setTileGID:GID3 at:ccp(x,_tileMap.mapSize.height - y
            - 1)];
        [_background setTileGID:GID3 at:ccp(x,_tileMap.mapSize.height - y
            + 1)];

        if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp(x,
            _tileMap.mapSize.height - y - 3)]][@"Collidable"]
            isEqualToString:@"False"])
        {
            [_background setTileGID:GID2 at:ccp(x,_tileMap.mapSize.height
                - y - 2)];
        }
        if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp(x,
            _tileMap.mapSize.height - y + 3)]][@"Collidable"]
            isEqualToString:@"False"])
        {
            [_background setTileGID:GID2 at:ccp(x,_tileMap.mapSize.height
                - y + 2)];
        }
    }

    x = Node.split.intValue - 1;
    //propogate path to the left
    for (;;x--)
    {
        if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp(x,
            _tileMap.mapSize.height - y)]]
            [@"Collidable"]isEqualToString:@"False"])
        {
            for (int tempX = x;tempX>x-3;tempX--)
            {
                [_background setTileGID:GID2 at:ccp(tempX,_tileMap.mapSize
                    .height - y)];
                [_background setTileGID:GID3 at:ccp(tempX,_tileMap.mapSize
                    .height - y - 1)];
                [_background setTileGID:GID3 at:ccp(tempX,_tileMap.mapSize
                    .height - y + 1)];
                if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp
                    (tempX,_tileMap.mapSize.height - y - 3)]][@"Collidable"
```

```
                    ]isEqualToString:@"False"])
                {
                    [_background setTileGID:GID2 at:ccp(x,_tileMap.mapSize
                        .height - y - 2)];
                }
                if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp
                    (tempX,_tileMap.mapSize.height - y + 3)]][@"Collidable"
                    ]isEqualToString:@"False"])
                {
                    [_background setTileGID:GID2 at:ccp(x,_tileMap.mapSize
                        .height - y + 2)];
                }
            }
            break;
        }

        [_background setTileGID:GID2 at:ccp(x,_tileMap.mapSize.height - y)
            ];
        [_background setTileGID:GID3 at:ccp(x,_tileMap.mapSize.height - y
            - 1)];
        [_background setTileGID:GID3 at:ccp(x,_tileMap.mapSize.height - y
            + 1)];

        if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp(x,
            _tileMap.mapSize.height - y - 3)]][@"Collidable"]
            isEqualToString:@"False"])
        {
            [_background setTileGID:GID2 at:ccp(x,_tileMap.mapSize.height
                - y - 2)];
        }
        if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp(x,
            _tileMap.mapSize.height - y + 3)]][@"Collidable"]
            isEqualToString:@"False"])
        {
            [_background setTileGID:GID2 at:ccp(x,_tileMap.mapSize.height
                - y + 2)];
        }
    }
}
else
{
    //horizontal split
    int topx = MIN([self maxTopRightx:Node.leftNode]-1,[self maxTopRightx:
        Node.rightNode]-1);
    int botx = MAX([self minBotLeftx:Node.leftNode]+1,[self minBotLeftx:
        Node.rightNode]+1);
    float fraction = arc4random()%91;
    fraction = (fraction/100)+0.05;
    fraction = (fraction*(topx - botx))+botx;
    int x = fraction;
    int y = Node.split.intValue;
    //propogate path upwards
    for (;;y++)
    {
        if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp(x,
            _tileMap.mapSize.height - y)]]
            [@"Collidable"]isEqualToString:@"False"])
        {
            for (int tempY = y;tempY<y+3;tempY++)
            {
```

```
            if (tempY%2)
            {
                [_background setTileGID:GID+14 at:ccp(x,_tileMap.
                    mapSize.height - tempY)];
                [_background setTileGID:GID+14 at:ccp(x-1,_tileMap.
                    mapSize.height - tempY)];
                [_background setTileGID:GID+14 at:ccp(x+1,_tileMap.
                    mapSize.height - tempY)];
                if ([[_tileMap propertiesForGID:[_background
                    tileGIDAt:ccp(x-3,_tileMap.mapSize.height - tempY)]
                    ][@"Collidable"]isEqualToString:@"False"])
                {
                    [_background setTileGID:GID+14 at:ccp(x-2,_tileMap
                        .mapSize.height - y)];
                }
                if ([[_tileMap propertiesForGID:[_background
                    tileGIDAt:ccp(x+3,_tileMap.mapSize.height - tempY)]
                    ][@"Collidable"]isEqualToString:@"False"])
                {
                    [_background setTileGID:GID+14 at:ccp(x+2,_tileMap
                        .mapSize.height - y)];
                }
            }
            else
            {
                [_background setTileGID:GID+16 at:ccp(x,_tileMap.
                    mapSize.height - tempY)];
                [_background setTileGID:GID+16 at:ccp(x-1,_tileMap.
                    mapSize.height - tempY)];
                [_background setTileGID:GID+16 at:ccp(x+1,_tileMap.
                    mapSize.height - tempY)];
                if ([[_tileMap propertiesForGID:[_background
                    tileGIDAt:ccp(x-3,_tileMap.mapSize.height - tempY)]
                    ][@"Collidable"]isEqualToString:@"False"])
                {
                    [_background setTileGID:GID+16 at:ccp(x-2,_tileMap
                        .mapSize.height - y)];
                }
                if ([[_tileMap propertiesForGID:[_background
                    tileGIDAt:ccp(x+3,_tileMap.mapSize.height - tempY)]
                    ][@"Collidable"]isEqualToString:@"False"])
                {
                    [_background setTileGID:GID+16 at:ccp(x+2,_tileMap
                        .mapSize.height - y)];
                }
            }
        }
        break;
    }
    if (y%2)
    {
        [_background setTileGID:GID+14 at:ccp(x,_tileMap.mapSize.
            height - y)];
        [_background setTileGID:GID+14 at:ccp(x-1,_tileMap.mapSize.
            height - y)];
        [_background setTileGID:GID+14 at:ccp(x+1,_tileMap.mapSize.
            height - y)];
        if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp(x-3
            ,_tileMap.mapSize.height - y)]][@"Collidable"]
            isEqualToString:@"False"])
```

```
            {
                [_background setTileGID:GID+14 at:ccp(x-2,_tileMap.mapSize
                    .height - y)];
            }
            if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp(x+3
                ,_tileMap.mapSize.height - y)]][@"Collidable"]
                isEqualToString:@"False"])
            {
                [_background setTileGID:GID+14 at:ccp(x+2,_tileMap.mapSize
                    .height - y)];
            }
        }
        else
        {
            [_background setTileGID:GID+16 at:ccp(x,_tileMap.mapSize.
                height - y)];
            [_background setTileGID:GID+16 at:ccp(x-1,_tileMap.mapSize.
                height - y)];
            [_background setTileGID:GID+16 at:ccp(x+1,_tileMap.mapSize.
                height - y)];
            if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp(x-3
                ,_tileMap.mapSize.height - y)]][@"Collidable"]
                isEqualToString:@"False"])
            {
                [_background setTileGID:GID+16 at:ccp(x-2,_tileMap.mapSize
                    .height - y)];
            }
            if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp(x+3
                ,_tileMap.mapSize.height - y)]][@"Collidable"]
                isEqualToString:@"False"])
            {
                [_background setTileGID:GID+16 at:ccp(x+2,_tileMap.mapSize
                    .height - y)];
            }
        }
    }

    y = Node.split.intValue-1;
    //propogate path downwards
    for (;;y--)
    {
        if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp(x,
            _tileMap.mapSize.height - y)]]
            [@"Collidable"]isEqualToString:@"False"])
        {
            for (int tempY = y;tempY>y-3;tempY--)
            {
                if (tempY%2)
                {
                    [_background setTileGID:GID+14 at:ccp(x,_tileMap.
                        mapSize.height - tempY)];
                    [_background setTileGID:GID+14 at:ccp(x-1,_tileMap.
                        mapSize.height - tempY)];
                    [_background setTileGID:GID+14 at:ccp(x+1,_tileMap.
                        mapSize.height - tempY)];
                }
                else
                {
                    [_background setTileGID:GID+16 at:ccp(x,_tileMap.
                        mapSize.height - tempY)];
```

```
                        [_background setTileGID:GID+16 at:ccp(x-1,_tileMap.
                            mapSize.height - tempY)];
                        [_background setTileGID:GID+16 at:ccp(x+1,_tileMap.
                            mapSize.height - tempY)];
                    }
                }
                break;
            }

            if (y%2)
            {
                [_background setTileGID:GID+14 at:ccp(x,_tileMap.mapSize.
                    height - y)];
                [_background setTileGID:GID+14 at:ccp(x-1,_tileMap.mapSize.
                    height - y)];
                [_background setTileGID:GID+14 at:ccp(x+1,_tileMap.mapSize.
                    height - y)];
                if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp(x-3
                    ,_tileMap.mapSize.height - y)]][@"Collidable"]
                    isEqualToString:@"False"])
                {
                    [_background setTileGID:GID+14 at:ccp(x-2,_tileMap.mapSize
                        .height - y)];
                }
                if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp(x+3
                    ,_tileMap.mapSize.height - y)]][@"Collidable"]
                    isEqualToString:@"False"])
                {
                    [_background setTileGID:GID+14 at:ccp(x+2,_tileMap.mapSize
                        .height - y)];
                }
            }
            else
            {
                [_background setTileGID:GID+16 at:ccp(x,_tileMap.mapSize.
                    height - y)];
                [_background setTileGID:GID+16 at:ccp(x-1,_tileMap.mapSize.
                    height - y)];
                [_background setTileGID:GID+16 at:ccp(x+1,_tileMap.mapSize.
                    height - y)];
                if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp(x-3
                    ,_tileMap.mapSize.height - y)]][@"Collidable"]
                    isEqualToString:@"False"])
                {
                    [_background setTileGID:GID+16 at:ccp(x-2,_tileMap.mapSize
                        .height - y)];
                }
                if ([[_tileMap propertiesForGID:[_background tileGIDAt:ccp(x+3
                    ,_tileMap.mapSize.height - y)]][@"Collidable"]
                    isEqualToString:@"False"])
                {
                    [_background setTileGID:GID+16 at:ccp(x+2,_tileMap.mapSize
                        .height - y)];
                }
            }
        }
    }
}

- (int) maxTopRighty:(BSPNode*)Node
```

```
{
    //return the maximum top right corner y coordinate of the node and it's
        child nodes
    if (Node.leftNode)
    {
        int y1 = [self maxTopRighty:Node.leftNode];
        int y2 = [self maxTopRighty:Node.rightNode];
        return MAX(y1,y2);
    }
    else
    {
        return Node.topRighty.intValue;
    }
}

- (int) minBotLefty:(BSPNode*)Node
{
    //return the minimum bottom left corner y coordinate of the node and it's
        child nodes
    if (Node.leftNode)
    {
        int y1 = [self minBotLefty:Node.leftNode];
        int y2 = [self minBotLefty:Node.rightNode];
        return MIN(y1,y2);
    }
    else
    {
        return Node.botLefty.intValue;
    }
}

- (int) maxTopRightx:(BSPNode*)Node
{
    //return the maximum top right corner x coordinate of the node and it's
        child nodes
    if (Node.leftNode)
    {
        int y1 = [self maxTopRightx:Node.leftNode];
        int y2 = [self maxTopRightx:Node.rightNode];
        return MAX(y1,y2);
    }
    else
    {
        return Node.topRightx.intValue;
    }
}

- (int) minBotLeftx:(BSPNode*)Node
{
    //return the minimum bottom left corner x coordinate of the node and it's
        child nodes
    if (Node.leftNode)
    {
        int y1 = [self minBotLeftx:Node.leftNode];
        int y2 = [self minBotLeftx:Node.rightNode];
        return MIN(y1,y2);
    }
    else
    {
        return Node.botLeftx.intValue;
```

```
    }
}

- (void) spawnEnemies:(BSPNode*)Node
{
    //create enemies in the rooms of the map
    if (Node.leftNode)
    {
        [self spawnEnemies:Node.leftNode];
        [self spawnEnemies:Node.rightNode];
    }
    else
    {
        int spaceX = (Node.topRightx.intValue - Node.botLeftx.intValue)/8;
        int spaceY = (Node.topRighty.intValue - Node.botLefty.intValue)/8;
        for (int x = 0; x < spaceX; x++)
        {
            for (int y = 0; y < spaceY; y++)
            {
                if (abs((Node.botLeftx.intValue + 4 + (x*8))*8 - _player.
                    position.x) > 60 && abs((Node.botLefty.intValue + 4 + (y*8)
                    )*8 - _player.position.y) > 60)
                {
                    int spawn = arc4random()%101;
                    int threshold;
                    if ([[Engine sharedInstance] currentLevel].intValue > 15)
                    {
                        threshold = 15;
                    }
                    else
                    {
                        threshold =[[Engine sharedInstance] currentLevel].
                            intValue;
                    }
                    if (spawn > 40 - threshold)
                    {
                        //spawn enemy
                        Enemy *enemy = [Enemy
                            spriteWithFile:@"enemynatural.png"];
                        enemy.position = ccp((Node.botLeftx.intValue + 4 + (x*
                            8))*8,(Node.botLefty.intValue + 4 + (y*8))*8);
                        [enemy initWithHealth:100 + ([[Engine sharedInstance]
                            currentLevel].intValue*100) Damage:1];
                        enemy.rotation = arc4random()%361;
                        [self addChild:enemy z:1];
                        [self addChild:[enemy healthDisplayReference] z:1];

                        NSMutableArray *objectsArray;

                        if (_enemys != nil)
                        {
                            objectsArray = [NSMutableArray arrayWithArray:
                                _enemys.allObjects];
                            [objectsArray addObject:enemy];
                            _enemys = [NSMutableSet setWithArray:objectsArray]
                                ;
                        }
                        else
                        {
                            _enemys = [NSMutableSet setWithObject:enemy];
```

```
                        }
                    }
                }
            }
        }
    }

- (void) setExit:(BSPNode*)Node
{
    //set the position of the exit of the map
    if (Node.leftNode)
    {
        BOOL left = arc4random()%2;
        if (left)
        {
            [self setExit:Node.leftNode];
        }
        else
        {
            [self setExit:Node.rightNode];
        }
    }
    else
    {
        if (((Node.topRightx.intValue + Node.botLeftx.intValue)/2)*8 ==
            _player.position.x && ((Node.topRighty.intValue + Node.botLefty.
            intValue)/2)*8 == _player.position.y)
        {
            [self setExit:_topNode];
        }
        else
        {
            int GID = _background.tileset.firstGid;
            [_background setTileGID:GID+37 at:ccp((Node.topRightx.intValue +
                Node.botLeftx.intValue)/2 - 1,_tileMap.mapSize.height - (Node.
                topRighty.intValue + Node.botLefty.intValue)/2)];
            [_background setTileGID:GID+38 at:ccp((Node.topRightx.intValue +
                Node.botLeftx.intValue)/2,_tileMap.mapSize.height - (Node.
                topRighty.intValue + Node.botLefty.intValue)/2)];
            [_background setTileGID:GID+39 at:ccp((Node.topRightx.intValue +
                Node.botLeftx.intValue)/2 - 1,_tileMap.mapSize.height - (Node.
                topRighty.intValue + Node.botLefty.intValue)/2 - 1)];
            [_background setTileGID:GID+40 at:ccp((Node.topRightx.intValue +
                Node.botLeftx.intValue)/2,_tileMap.mapSize.height - (Node.
                topRighty.intValue + Node.botLefty.intValue)/2 - 1)];
        }
    }
}

-(void) died
{
    //player has died
    [[Engine sharedInstance] setPausedVariable: [NSNumber numberWithBool:1]];
    [[Engine sharedInstance] setAlive:[NSNumber numberWithBool:0]];

    for (Enemy *enemy in _enemys.allObjects)
    {
        if (enemy != nil)
        {
```

```
            [self removeChild:enemy cleanup:YES];
            [self removeChild:enemy.healthDisplayReference cleanup:YES];
        }
    }
    _enemys = nil;
    _shadows = [_shadowMap layerNamed:@"Shadows"];
    [_shadows setTileGID:(_shadows.tileset.firstGid + 1) at:ccp(_player.
        position.x/8,_tileMap.mapSize.height - (_player.position.y/8))];
}

- (void) save
{
    //save the map and enemies
    NSMutableArray *tiles = [[NSMutableArray alloc]init];
    for (int x = 200; x < 300; x++)
    {
        for (int y = 200; y < 300; y++)
        {
            [tiles addObject:[NSNumber numberWithInt:[_background tileGIDAt:
                ccp(x,y)]]];
        }
    }

    int n = 0;
    NSMutableArray *enemies = [[NSMutableArray alloc]init];
    for (Enemy *object in _enemys.allObjects)
    {
        if (object != nil)
        {
            [enemies addObject:[NSNumber numberWithInt:object.position.x]];
            [enemies addObject:[NSNumber numberWithInt:object.position.y]];
            [enemies addObject:[NSNumber numberWithInt:object.health.intValue]
                ];
            [enemies addObject:[NSNumber numberWithInt:object.activated.
                intValue]];
            [enemies addObject:[NSNumber numberWithFloat:object.rotation]];
            n++;
        }
    }

    NSDictionary *dictionary = [NSDictionary dictionaryWithObjectsAndKeys:
                                tiles, @"tiles",
                                enemies, @"enemies", nil];
    NSString *rootPath = [NSSearchPathForDirectoriesInDomains
        (NSDocumentDirectory, NSUserDomainMask, YES) objectAtIndex:0];
    NSString *filePath = [rootPath stringByAppendingPathComponent:@"appData"];
    [NSKeyedArchiver archiveRootObject:dictionary toFile:filePath];
    [[Engine sharedInstance] setSaved:[NSNumber numberWithBool:true]];
}

- (void) onExitTransitionDidStart
{
    //run the save algorithm when the layer is closing
    [self save];
}

- (void) onEnter
{
    //set up the map after the layer appears
    [super onEnter];
```

```
    if ([[Engine sharedInstance] startingNewGame].boolValue)
    {
        for (int x = 200; x <=300; x++)
        {
            for (int y = 200; y <=300; y++)
            {
                [_background setTileGID:(_background.tileset.firstGid) at:ccp
                    (x,y)];
                [_shadows setTileGID:(_shadows.tileset.firstGid) at:ccp(x,y)];
            }
        }
        for (Enemy *enemy in _enemys.allObjects)
        {
            if (enemy != nil)
            {
                [self removeChild:enemy cleanup:YES];
                [self removeChild:enemy.healthDisplayReference cleanup:YES];
            }
        }
        _enemys = nil;
        [self constructMap];
        int GID = _shadows.tileset.firstGid + 1;
        [_shadows setTileGID:GID at:ccp(_player.position.x/8,_tileMap.mapSize.
            height - (_player.position.y/8))];
    }
}

@end
```

Permission to reproduce all copyright materials have been applied for. In some cases, efforts to contact copyright holders have been unsuccessful and AQA will be happy to rectify any omissions of acknowledgements in future documents if required.