IT 2931 Game Development Techniques

Mikonos Studio

Technical Document

Game Title:	Dungeon Revival		
Company name:	Mikonos		
Company Website:	http://www.geocities.com/hkzr/Mikonos		
Team Member:	Hansel Koh	050664U	(Project leader)
	Chen Ren Hao	052663Z	(Audio Person + Marketing)
	Nigel Low Wei Yang	052365B	(Lead programmer + Tester)
	Melissa Lim	054131H	(Designer + Artist)

Content

Content

Game functionalities

Main Menu

- **Basic background and buttons**

  Inputs: Default background location and number of buttons. File name of textures for all the quads.

  Function: Interface for the game state machine

  Returns: ~

  Process: Input is checked to see which selection is selected.

- **Mouse motion and button functionalities**

  Inputs: XY coordinates and clicks of the mouse from the relevant OpenGL callback functions. Current game state.

  Function: Interface for the user's input.

  Returns: Mouse location upon clicking or motion of the mouse. (Depending on the game state)

  Process: Varies widely with current state of game.

- **Customization (Customize Settings)**

  Inputs: Mouse location upon clicking.

  Function: Interface to change user's preferences.

  Returns: Game volume, Mouse sensitivity and keyboard left-right-forward-back configurations.

  Process: Input is checked to see which selection is selected.

- **Credits**

  Inputs: ~

  Function: Interface to display the people behind the game.

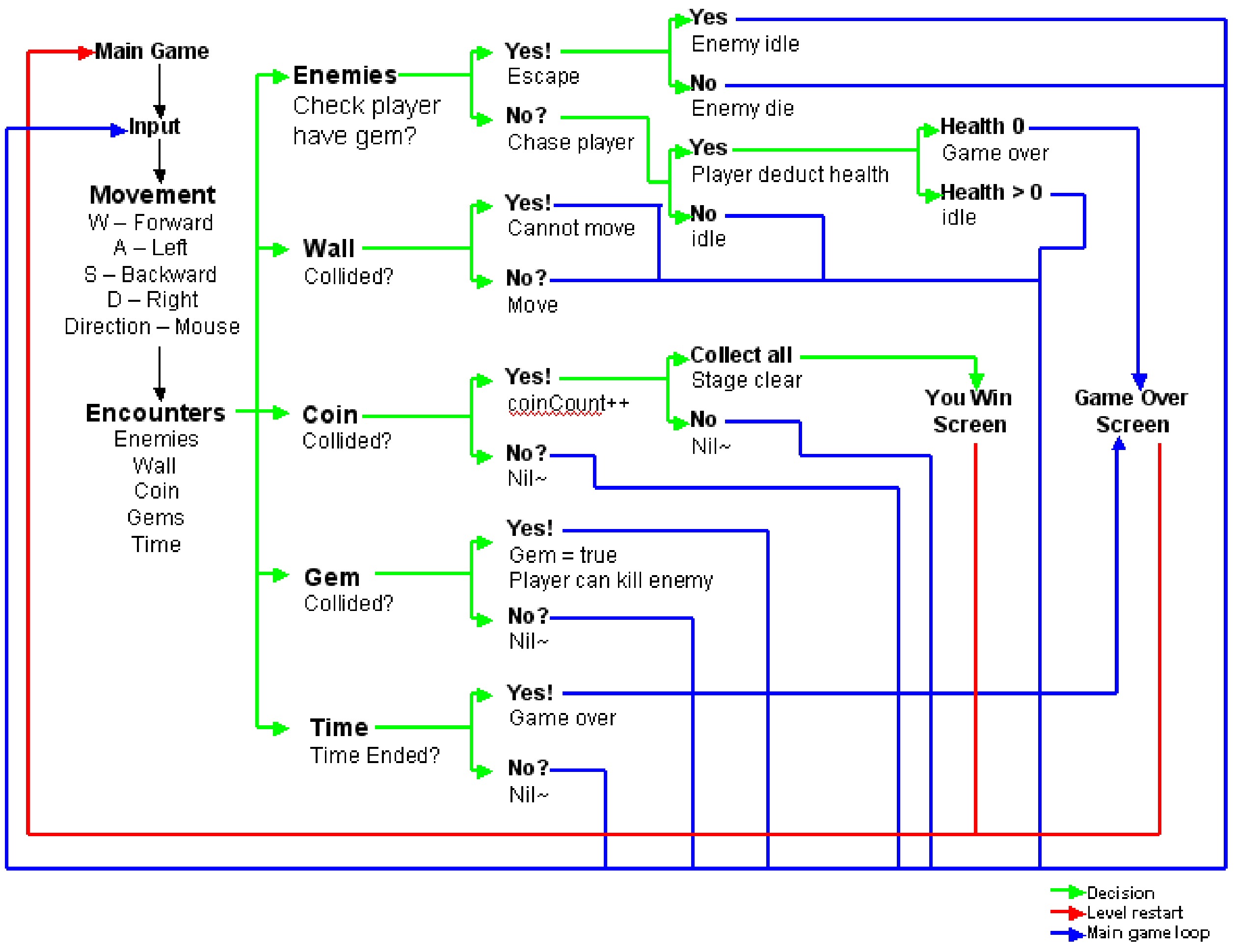
  Returns: ~

  Process: Animated scrolling text

Game functionalities

Mouse and keyboard input

- **Input flow chart**



Game functionalities

Mouse and keyboard input

- **Mouse movement detection and mouse-clicks**

Inputs:	Mouse cursor location coordinates (X,Y) Mouse button's ID upon clicking State of button (1 or 0)
Function:	The mouse listener in AppMain. Passes on mouse data to the necessary functions of the classes depending on the gamestate
Returns:	~
Process:	Checks the gamestate, then calls the mouse listener in the appropriate class.

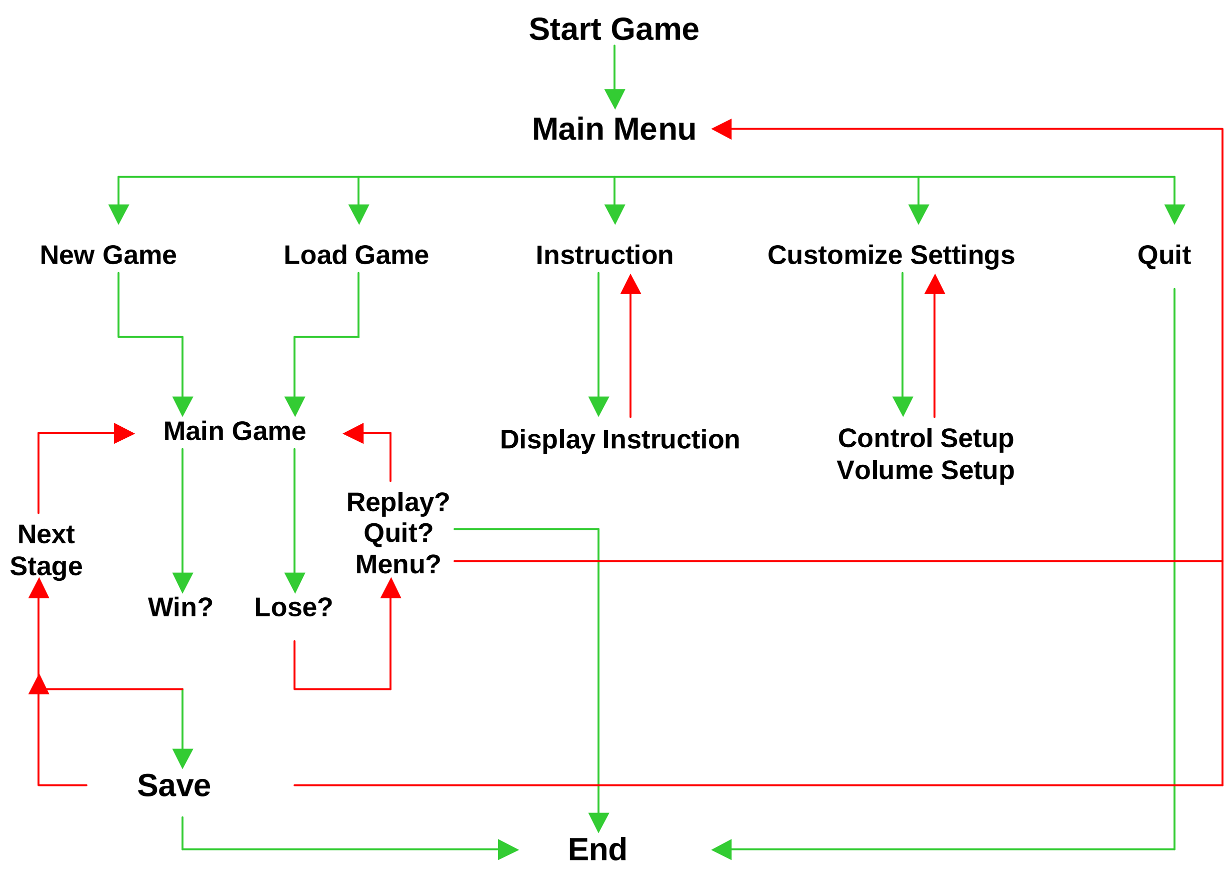
- **Keyboard key-inputs**

Inputs:	Unsigned char of key pressed and released by the user.
Function:	The key listener in AppMain. Passes on data to the necessary functions of the classes depending on the gamestate.
Returns:	~
Process:	Checks the gamestate, then calls the keyboard listener in the appropriate class.

Game functionalities

Game state machine

- **Game flow chart**



Game functionalities

Game state machine

- **Menu states**

Inputs:	The states where the menu function are in like gameworld, load game, instructions, settings, credits, quit game.
Function:	To process, listening to key input, mouse events, and render screen based on the state of the menu.
Returns:	The things to process like key and mouse input, rendering information.
Process:	Render a screen that needs to be rendered, passing in of key and mouse input to various functions.

- **Instructions states**

Inputs:	The page where the instructions is at.
Function:	Track the instruction page where player is viewing.
Returns:	The things to process like rendering information.
Process:	Render a screen that needs to be rendered.

- **Customize states**

Inputs:	The volume, mute to max. The mouse sensitive, low to high The directional keys, forward, left, back, right. The default settings.
Function:	Letting player to set their own preference of control of the game and will be save after they are set.
Returns:	Save whatever the player have set in the customize settings.
Process:	Apply whatever the player have set in the customize settings to the game play.

Game functionalities

Game state machine

- **AI or Monster states**

Inputs:	Idle. Player have gem. Chase player. Return to original position.	
Function:	Idle	– not doing anything.
	Player have gem	– run away from player
	Chase player	– chase after player when he got no gem.
	Return to original location	– When player in not in sight.
Returns:	Actions of the monster.	
Process:	Calling various functions that will execute various types of actions.	

- **Win and lose splash screen and scoreboard**

Inputs:	Number of coins and health.
Function:	If health is 0 will lose state. If coin count is same as the coin total will win state.
Returns:	The state of the game whether it is loses or wins.
Process:	Render win screen or the lose screen.

Game functionalities

Dynamic game level

- **3D array grid-based game level**

Inputs:	Pass in the text file of the map that will be loaded. And store them in the array that will then later be used to draw map, check collision and path finding.
Function:	Keep tracks of the items in the game world.
Returns:	Items that are in the game world.
Process:	Text file -> Read-in -> Store

- **Coins, Gems and Monster locations**

Inputs:	Each items walls, coins and gems are represented with a number.
Function:	It's for easy level building in future and able to design out from the map editor that is created in flash.
Returns:	Draw out various items in their desire locations.
Process:	Planed -> Store -> Draw

- **Flash map-editor**

Inputs:	The location and types of walls, gems and coins in the maze or map of the game.
Function:	Created in flash to help map designer to see the layout of maps making it easy for them to design the map.
Returns:	After the map has been design, a list of number will then be generated. All this number will then refer to the type of walls, gems and coins locations.
Process:	It will convert the design of the map in the map editor in to arrays of numbers and then later will then be read in by the map loader in the game and will then draw out the map based on the numbers generated from the map designer in the notepad.

Game functionalities

Object and texture loaders

- **Game objects and material**

Inputs:	File name and location of the .tml and .obj files
Function:	To load and render out the object from the .obj file and apply materials to it.
Returns:	When render function is called, object is displayed.
Process:	Receive and store the filename when constructor is called.

- **Texture (.tga)**

Inputs:	File name and location of the .tga files, and ID to be assigned to the texture.
Function:	Assigns a texture's data into an array of textures for the game
Returns:	When the bind-texture function is called, the corresponding texture from the array is assigned to every vertices thereafter until texture-binding is disabled.
Process:	Target texture file is loaded, converted into byte data, mipmap and texture parameters are created and passed into the texture array and assigned the specified ID.

Game functionalities

Camera

- **Camera states**

Inputs:	Game State
Function:	Sets the camera correctly for the different states of the game.
Returns:	Sets the gluLookAt function's parameters.
Process:	Checks the game state and sets the camera parameters accordingly.

- **Spring Damping**

Inputs:	Current Position - position of the camera Target Position - position the camera should go to Previous Target Position - position the camera should previously go to Change in Time - To factor in time Hooke's Constant - the amount to spring Damp Constant - the acceleration SpringLen - average displacement from target position
Function:	To add realism to the camera's movement
Returns:	An updated position of the camera
Process:	A vector is calculated out using all the inputs and returned to the invoker.

Game functionalities

Sound

- **Main Menu**

Inputs:	Value of the volume set by customizes setting.
Function:	Set the volume specified in customizes setting.
Returns:	~
Process:	FSOUND_SetSFXMasterVolume will set the volume of all the sounds and music in the game.

- **Game Level (all Gameplay sounds)**

Inputs:	Boolean to set whether to play the incidental sounds.
Function:	Play the different type of incidental sounds.
Returns:	~
Process:	FSOUND_SAMPLE will decompress the incidental samples into memory. FSOUND_PlaySound will play the sample in its function using any free channel available.

- **Background soundtracks**

Inputs:	Boolean to set whether to play the background music.
Function:	Play the different type of background music.
Returns:	~
Process:	FSOUND_STREAM will stream in the background music. FSOUND_Stream_Play will play the background music in its function using any free channel available.

Game functionalities

Collision detection

- **Enemies**

Inputs:	Information from a text file, state of A.I and location of player.
Function:	Determines whether monster has caught the player or if player has gem, whether the player has killed the enemy.
Returns:	If player is within range, if player is in direct contact and is this enemy Dead or Alive.
Process:	Checks with its collision box and the player's to determine if in range or in direct contact.

- **Walls**

Inputs:	Location of player.
Function:	Determines whether player has reached an impassable area. Bounces the player back if in direct contact.
Returns:	~
Process:	Checks with the walls location and the player's to determine if in direct contact.

- **Coins and Gems**

Inputs:	Location of player.
Function:	Determines whether player is in direct contact, if true, coin/gem is collected and appropriate effects are applied.
Returns:	~
Process:	Checks with the object locations and the player's collision box to determine if in direct contact.

Game functionalities

Mini-map and Heads-up-Display

- **Mini-map level layout and Character position**

Inputs:	Player's position and angle, enemies' position and angle, type of wall information.
Function:	Display the player, enemies and the level layout on the mini map.
Returns:	~
Process:	The player's position and angle and enemies' position and angle will keep updating. These positions will be used to translate the triangles in the mini map. The angle is use to rotate the triangles which indicate the direction in which the character is facing. The type of wall information is used to draw a smaller version of the type of wall in the mini map.

A.I. finite state machine and path finding

- **Finite State Machine and A.I. flow chart**

Inputs:	Player's position Presence of gem on the player
Function:	Controls the enemies' response to the player
Returns:	Calls the necessary functions and changes state if necessary.
Process:	State of A.I. is checked and all necessary responses and their conditions are calculated.

- **Path-finding**

Inputs:	Position of invoker. Position of target (not necessary always the player).
Function:	A series of function to move the monster as a response to the player.
Returns:	Updates the monster's position
Process:	First the target position is checked to see if its visible by testing whether a straight line to the target gets intersected by walls. Then the position of the monster is updated to approach the position.

Task allocation list

Project leader: Hansel Koh

Started off with discussion on what are the possible games that can be developed within the time frame of 17 weeks. And with ideas and concepts that are been brainstorm by the team, more meetings are held to discus the assignments of jobs, then how and which method can be used and what cannot be use in the development. After that, will be the problems that the team will be facing in future along development and what are the possible ways to avoid or over come the problems.

After planning stage is over, started design the game world and decided to use a grid system with a map editor that is created in flash 8, so making it simple to design new map as well as build levels. The concept was simple, that is to use flash to design the map and generate a array of information on the map, save it to .txt file then the game will read in and store the information in to a 3D array, and then render the map on to the game world.

Next will then to add gems and coins in to the game world. The location of the gems and coins are designed in the map editor and export to .txt file same method used for wall.

Create movement for the main character to navigate around the world using the standard “a”, “s”, “d”, and “w” key control. Also setting the mouse cursor to the center of the screen in alternate frame, as to allow the character can turn using mouse control.

With the wall, gem and coin implemented in the game world, collision detection is then added into the game using the position of the character and the top left and bottom right coordinates of wall to check whether the character is in the wall, and make a bounce if collide with wall. And collision of coin and gem will be using the position of the coin or gem with the character’s position. When collisions are working, will be to make the response of the collision such as make coin and gem disappear when taken and health drop when monster touch character.

A status screen is then created to display the number of coin, gem and health of the character by using orthogonal matrix mode to render a 2D screen over the game world.

Then further complete states of the game such as win or lose screen based on the health or coins collected and A.I. states to control behavior of monster when character is within range and does character have a gem.

Task allocation list

Audio Person + Marketing: Chen Ren Hao

At the beginning of the semester, I begin my first assignment by researching on the implementation of FMOD for playing sounds and music. After a few days of research, I am able to get FMOD working and able to play simple background music for our menu and game world.

As there is no game play at the beginning, the next thing I worked on is the customize setting. The customize setting allows the player to adjust the volume of background music and sounds, set the mouse sensitivity and set the keys for the movement of the character. There is also a default button if the player does not want to adjust anything. All the information is written to a text file to save the data.

After completing the customize setting, I start to work on the mini map for our game. The mini map is drawn using orthogonal. First, I get the position vector of the character and get it to translate a quad on the mini map. After I manage to do this, the next thing is to get the information of the wall and draw it on the mini map. The information of the wall is store in a 3x3 array. Having that information, I draw a small version of the different type of walls on the mini map. After drawing of the wall, I add the direction of the character on the mini map. I make use of the direction vector of the character and reflect it on the mini map. The last thing I implement on the map is adding the monster in the mini map when the character is close to the monster.

After I finished the mini map, I work back on the sound again. I start to add in incidental sounds and also change the background music at different state of the game. I also create a class for the sounds. This will tidy up the code.

Next I implement the time for taking the gem. The gem effect has different time limit. The time limit is random. I also work on the special effect when the character acquires a gem and coins. I make the particles rotate around the character when acquire the gem and randomly translate the particles up when acquire a coin. The next special effect I did was the effect for the monster when it dies. I gradually alpha out the monster and translate it up.

After working on the special effects, I put in some visual effect for taking damage. I made the character turn red for a while after attacked by the monster. I also solve some problems for the sounds after path finding has been implemented in our game.

Task allocation list

Lead programmer + Tester: Nigel Low Wei Yang

Menu: At the beginning of the semester and after basic game conceptualization, I started off the game by beginning the programming of a simple main menu. This menu displays a quad for the background and then creates many instances of the button class, which each of them handles its own rendering and animation, as well as its own position on the menu.

Mouse: After finalizing the Gameplay and the flow of the game with the group, and with the game-world up and running, I implemented mouse controls to the game. It behaves just like a standard shooter, where the mouse points the character in the direction and WASD keys to move. I store the location of the cursor and its previous location, find the difference and then apply it to the character's front facing vector, which then affects the side vector as well.

Camera: With the basic controls of the game up, I proceeded to make the camera. The basic camera was already done by Hansel, but I decided to use vectors to position and point the camera. This was to allow the camera to be modified to contain spring damping features. The spring damping algorithm was taken from the class' example code that demonstrates how it works and I took only the calculations and coded it into the camera. Some sticky and tricky problems were encountered but a few were solved.

AI path-finding: A few problems remained with the camera, but it was pretty much done with and the character didn't jerk as badly now, and thus I moved on to the hardest part, path-finding. After planning for one day I decided to break up this task into three steps where

1) Set-up the functions, in which vector leading to the player is calculated and the monster will follow this route.
2) The monster disengages when player is out of sight, which means being able to deduce whether the line to player is intersected by walls.
3) Finally, using the unplanned path for path-finding, go around the obstruction taking the nearest route (shortest path) until player is in sight. Once in sight, just charge at player.

Regrettably, I've only reached step two recently and step 3 is for future implementation. I met many hurdles in both step 1 and 2 and alas I have prevailed at the cost of time. Much time is used to solve each of the problems but much is learnt.

Administrative: As the lead programmer I did most of the documentation and did the version control. An ftp is used as a repository which members will take the game copy from and do their prototypes. I also did almost all of the comments for the codes and some of the framework.

Task allocation list

Designer + Artist: Melissa Lim

Started off with coming up with designs for the main character, and the storyline of the game. For the 1st few weeks, I concentrated on modeling the hero, monsters, gems and coins and doing the graphics of the game. I used Milkshape 3D as it was user friendly and allowed me to export the models into .obj format and automatically subdivide my quadrants into triangles. The models was plain initially, but limbs where added to the hero and monsters by week 13.

During the 5th week, I integrated Ms Dioselin's object loader codes into our main codes. After that, I went on to do a Credit screen for the game. I initialized another game state in the game and draw my Credit there. For the scrolling text, I incremented the count on y-axis until a certain number is reached, and then it will loop the animation again. I tried to synchronize the animation with the music.

The scoreboard came next, and I used the SetOrthoganal function for that game state, as the text was being rastered on screen. I read in the score (coin count and gem count) from status and print it in the scoreboard. The win and lose splash screen was also done in that period. I would read in the score and health, and if the score reached the max count, I would draw the win splash screen. If the health count is zero, I would draw the lose splash screen instead.

As the object loader only loads the materials of the models, our models do not have facial expression. Therefore, I added quadrants to the monsters as eyes, and mouth, and textured them accordingly. After that, I drew the shadows for the monster. As our game world is relatively bright, the shadow was small and faint.

Task allocation list

What went wrong?

1) The camera was jerking

2) Unable to formulate an algorithm to check if a line intersects another.

3) Wrong calculation of intersection of line.

4) The wall of the mini map is not drawn properly.

5) Unable to get the character on the mini map to rotate properly.

6) After path finding has been implemented, there are some bugs for the music. The background music did not play according to the state in the game.

7) Unclear definitions of the 3D array that keep track of map.

8) Adding more monsters to the game. Due the reading in of the monster location is in the monster class itself therefore problems occur when new monster class is been called.

9) Adding more monsters to the game. Due to the reading in function of the monster's data is within its class itself, problems occur when a new monster class is instantiated.

10) Game lags when whole game level is load.

11) Collision detection with walls, location of points wasn't clear.

12) Problems loading .mtl files.

13) Models without colors.

What went wrong?

Problem: The camera was jerking

Solution: This is a compound problem. It happens due to a myriad of reasons and they all point to the distance between character and camera being too unstable (literally jerking). First step was to set the character's keyboard movement to include the time factor as well, so that an element of speed shall reduce the jerkiness as the camera is moving to time. Second step was the mouse movement, its now also modified to include the time factor.

Unfortunately, the jerking is not fully resolved. Play the game a while and you will realize the character jerks occasionally (actually, rather frequently), unable to deduce that problem though.

Problem: Unable to formulate an algorithm to check if a line intersects another.

Solution: After trying to come up with an equation for a week, an algorithm found from the Internet was used instead but as we could not understand it we approached a teacher who explained it to us. Upon understanding it, this hurdle was done with.

Problem: Wrong calculation of intersection of line.

Solution: Discovered that the 3D array storing the map data is wrongly commented. Corrected the comments and updated the algorithm to take in the parameters in the correct order.

Problem: The wall of the mini map is not drawn properly.

Solution: The first 2 index of the 3D array need to switch to draw properly.

Problem: Unable to get the character on the mini map to rotate properly.

Solution: Make use of the yaw rotation value of the character in the game world and rotate accordingly in the mini map.

Task allocation list

Problems faced and solution

Problem: After path finding has been implemented, there are some bugs for the music. The background music did not play according to the state in the game. Example, the monster-chasing music is still playing after the character has collected a gem and also the monster-chasing music keep playing after the monster is dead.

Solution: Make a check for the AI state so as not to check when the monster is dead. This solves the problem for the monster-chasing music to keep playing after the monster is dead. Then stop playing the monster-chasing music after collected a gem.

Problem: Unclear definitions of the 3D array that keep track of map.

Solution: Re-Document the functions and stating each function of the array.

Problem: Adding more monsters to the game. Due the reading in of the monster location is in the monster class itself therefore problems occur when new monster class is been called.

Solution: For a temporary solution, monster class now takes in the location of monster and passed by parameters. But in the future, the reading in of location will be done outside of the monster class, making it more dynamic to create more monsters.

Problem: Lag of game when implemented full maze.

Solution: Used display list to store up the information of the maze and call the display list every time it render.

Problem: Collision detection with walls, location of points wasn't clear.

Solution: After re documented the functions of the array. And changed the array index.

Problem: Problems loading .mtl files.

Solution: The initial implementations of the mtl loader can't take in mtl files that are in a directory. Therefore then directories are then added into the code to help the mtl loader to find the mtl files that are required.

Problem: Models without colors.

Solution: Added light function to every model class to create a light on them to show the various materials color of the model.

Schedule

Milestones

1.) Set-up game skeleton :
- Main menu with working buttons
- Game-world with level loaded from a file
- Basic class methods of character and monster
- Finalize data structures
- Controls

2.) Main coding phrase : Implement features of the game :
- Camera
- Necessary models loaded
- Game-states and features for the menu

3.) Main coding phrase : Further implement features :
- Game flow
- Collision detection
- Sounds
- Heads Up Display (HUD)
- Finalize menu's features
- Advanced camera features

4.) Basic Gameplay. Game flow finalized and Gameplay elements are implemented:
- Win / lose game state
- Mini-map in HUD
- A.I. state
- Sound engine with dynamic background music
- Finalize all game models

5.) Going to version alpha. Complete all game features (Sounds and special effects), A.I. path-finding and create multiple monsters in game level.

Timeline

1st week <u>- Project assignment started</u>

- Project idea concepts

2nd week <u>- Project idea conceptualized</u>

- Design and create game level

3rd week <u>- End of conceptualizations, start of main coding phrase</u>

- Finalize all systems and create the basic classes for player and monsters
- Keyboard inputs detection
- Menu class for the game menu
- Code data structures (namely to store game level data in a 3D array from a file)

4th week **<u>- Reach first milestone and begin next step</u>**

- Basic camera and in-game controls
- Used a temporary model loader to display models
- Utilized mouse controls
- Final object loader being coded

5th week **<u>- Finalize game flow in menu and reach second milestone</u>**

- Optimized rendering of game-world to prevent lag during game
- Game states and game flow
- Modified camera to become a third person camera with use of vectors
- Begin work on spring damping camera
- Begin implementing sounds

7th week **<u>- Finalize game flow in menu and reach third milestone</u>**

- Basic Heads Up Display completed
- Collision detection with walls
- Implemented spring damped camera but with problems
- Collision detection for the game is documented
- Created a panel in the menu to allow players to set certain options

8th week **<u>- Finalize game flow in menu and reach third milestone</u>**

- Basic Gameplay up, with collision detection with Gameplay elements
- Added more options to the customize settings panel, and is now texture.

12th week **<u>- Finalize game flow in menu and reach third milestone</u>**

- Improved the collision detection
- A.I. state
- Textured character and monsters
- Begin work on path-finding
- Mini-map done for the HUD
- Credits screen in menu

13th week **<u>- Finalize game flow in menu and reach third milestone</u>**

- Models are now rendering correctly
- Monster data is now read in from a file
- First step of path-finding
- Special effects
- Scoreboard, win/lose screen for the game’s flow
- Models improved, instruction screen for game

14th week **<u>- Finalize game flow in menu and reach third milestone</u>**

- Monster faces the player
- Second step of path-finding
- Feedback when player is hit by monster
- Models improved

15th week **<u>- Project going to alpha version</u>**

- Prepare documentation and fully functional game