Alien Escape

Windows desktop game written in C++
using the Simple Direct Layer media library

Hons BSc. in Game Design and Development
Advanced Digital Game Programming

Game Design Document

David Morton K00179391

13.4.2015

Version 1.2

“You can run, but you'll die tired.”

Table of Contents

1. Game Design

1.1 Summary

Auron has escaped his cell and is running for his life, inside an alien hive, on an alien planet. He needs to stay alive long enough to find an escape or be rescued. Auron needs to dodge the aliens trying to capture him by leaping over them. Gravity is unstable so he needs to be prepared in case his world is turned upside down. Auron can boost to help him escape, until it overheats, and dies, typical.

1.2 Gameplay

The goal of the game is survival. The obstacles are aliens. The protagonist needs to dodge the obstacles in order to stay alive. If he gets hit too many times its game over. Gravity flips without warning causing the controls to change too. The player can boost to aid him in jumping over the enemies and avoid being hit. Use the arrow keys to jump up or down, depending on gravity and the space bar for boosting.

2. *Technical*

2.1 Scenes

Title Scene

- Background image
- Title (Centre Top)
- Start Button (Centre Bottom)

The title scene consists of a background image, a title label, and a start button with rendered text. The background image is a single PNG file rendered to an SDL texture. The start button is a UI element that can be selected. This will trigger a new game scene to be created. The world manager will destroy the menu scene and run the new game scene. The title “Alien Escape” shown in figure 1.0, is a static label with a rendered text string.

Figure 1.0: Title Scene and UI elements

Game Scene

- Background images
- Mid ground images
- Player sprite
- Enemy sprite

The game scene shown in figure 1.1 contains the player, enemy, background images, mid ground images. The background and mid ground images are PNG files rendered to an SDL texture. The background uses two images being scrolled side by side at a slower rate to the two mid ground images, causing a parallax effect. The player and enemy sprites are rendered above the background images.

HUD

- Distance score (Top left)
- Gravity Direction (Top Left)
- Lives Remaining (Bottom Right)
- Booster Status (Bottom Right)
- Pause Button (Top right)
- Debug FPS (Bottom Left)
- Debug Game Speed (Bottom)

The HUD consists of six labels with rendered text. There are two parts to each label, the text string and the value string. The value for each label is updated based on the player and game world statistics.

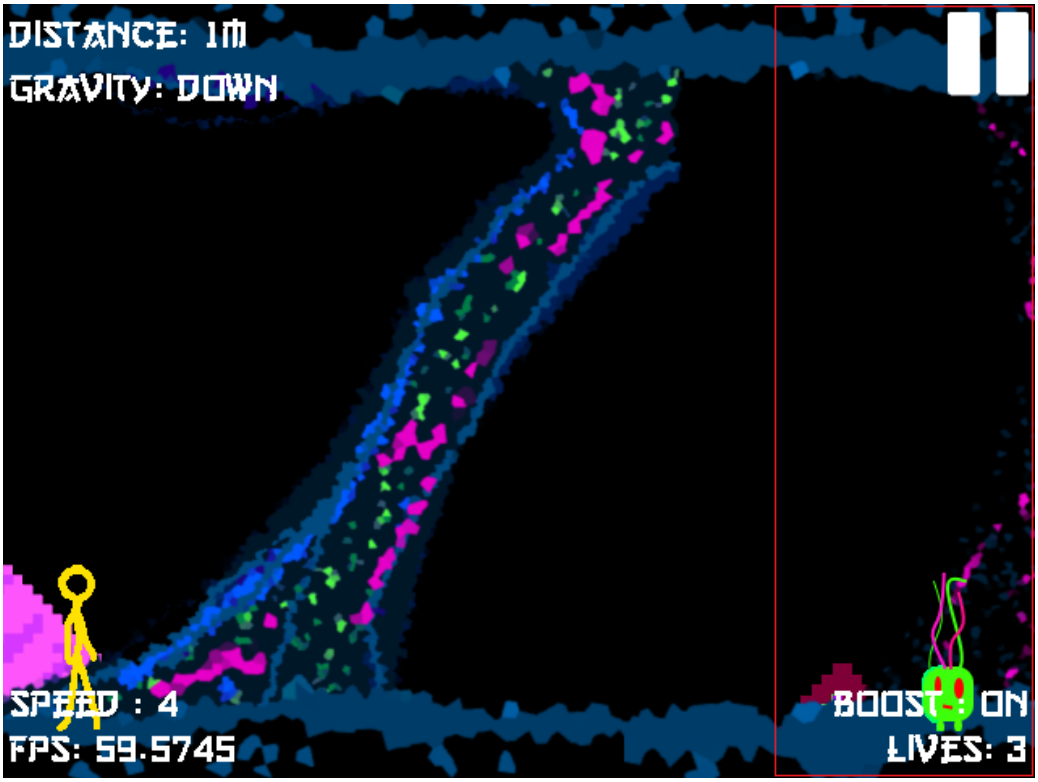

Figure 1.1: Game Scene and UI elements

2.2 Controls

- Press the Up key when gravity direction is DOWN to jump UP
- Press the Down key when gravity is UP to jump DOWN
- Confused?
- Press the spacebar to boost and increase your speed

Controls for the game consist of the up and down arrow keys, and the space bar. The mouse is used to interact with the start game button and the pause button. When gravity is down, the up key is used to control the player jump. When gravity is reversed the down key is used to control the player jump. Pressing down when gravity is up will cause the player to move downwards towards the floor. The space bar is used for boosting. This will increase the game and enemy speed, and will also increase the distance score

more often. Boost only lasts for 2 seconds with a 4 second cooldown. The space bar will not cause any effect on the game during the cooldown stage.

2.3 Mechanics

Physics will be implemented using a constant gravity value that will affect all objects in the game world. The direction of gravity will change depending on certain triggers in the game like a timer running out. Gravity will only flip when the enemy is within a safe zone. If the enemy is inside this safe zone show in figure 1.1 by the red rectangle, and the gravity timer reaches a certain value, gravity will flip. This will cause all game objects to be affected.

The player will run along the ground normally and when gravity flips he will run along the roof. The jump and boost function behaviours are relative to the players orientation. The enemy will jump when the player does but has a decreased velocity. This only allows the enemy to jump half as high as the player. The boost function will work the same for both orientations.

3. *Level Design*

3.1 Game Flow

Player starts in the alien hive running. Alien guards are chasing the player moving left from the right side of the screen. Player encounters an alien and the player must jump to avoid it. Gravity flips and the player runs on the roof. Gravity returns to normal and the player must jump to avoid aliens. Player boosts while jumping to aid evasion. Player gets hit and the game ends.

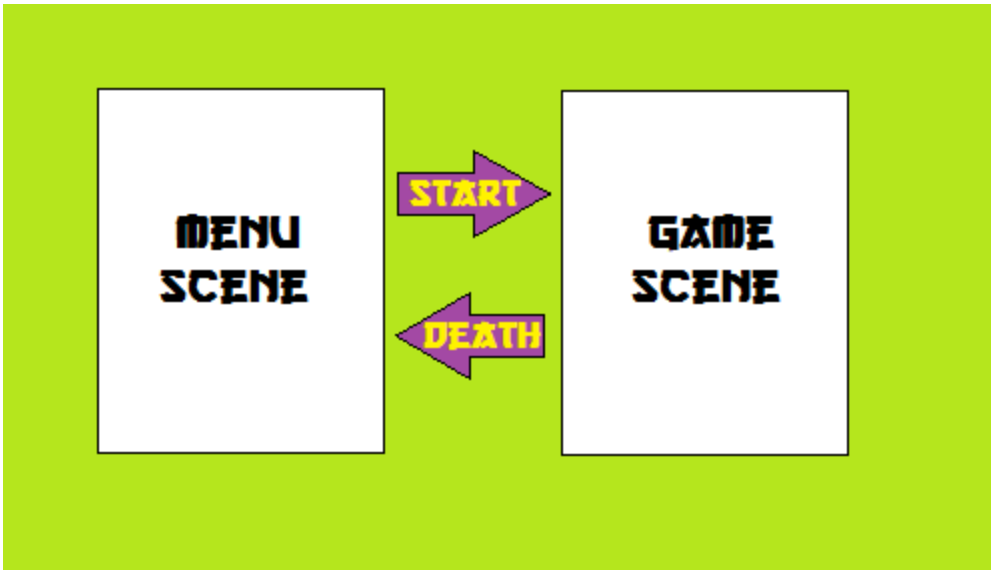

Figure 2.0: Scene transitions

The game starts in the menu scene and once the start button is selected the world manager runs the game scene. Figure 2.0 shows the transitions between both scenes. If the player dies the game scene ends and a new menu scene is loaded.

4. Development

4.1 Class Diagram

The class diagram shows all classes and inheritance for the game and the TinyXML2 library.

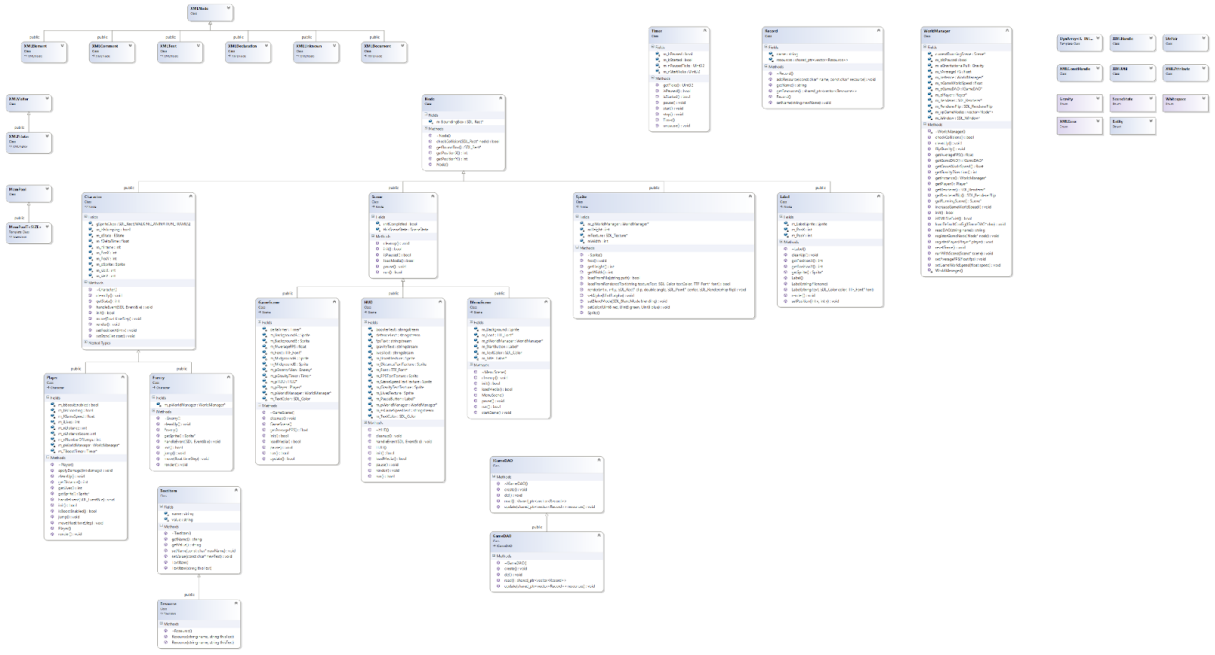

Figure 2.0: Alien Escape class diagram[1]

[1] https://raw.githubusercontent.com/damorton/AlienEscape/master/ClassDiagram.png

4.2 MVC Model View Controller

A MVC Model View Controller approach to design is used to structure the game. The user views the scenes that are being updated through the game loop. The World Manager controls the game world. The character update themselves based on user input.

Model

- Backgrounds
- Player
- Enemy
- HUD Labels

View

- Scene (Base Class)
- Menu Scene
- Game Scene
- Heads Up Display

Controller

- World Manager (Singleton)

Each section manages itself, the models update themselves, the views display the rendered game to the user, and the controller manipulates game world that the models are part of. The MVC approach is helpful for structuring game logic and control.

4.3 XML Data Access Object

The data access object is used to handle all communications between the game and its XML data storage file. The DAO consists of the following classes:

- IGameDAO
    - GameDao
- Record
- Text Item
    - Resource

The IGameDAO abstract class acts as an interface to the GameDAO object. The GameDAO is an XML implementation and is based on the CRUD design. Create, Read, Update, and Delete are the four main functions needed for the DAO. The Record class is used to group resources together for a particular section, scene, character, or object in the game. Each scene or game object in the game has a configuration record that contains resources like the file paths to the image files used for backgrounds, fonts, character sprites, and label text. Resources inherit data from the Text Item class that contains a string for a name and value.

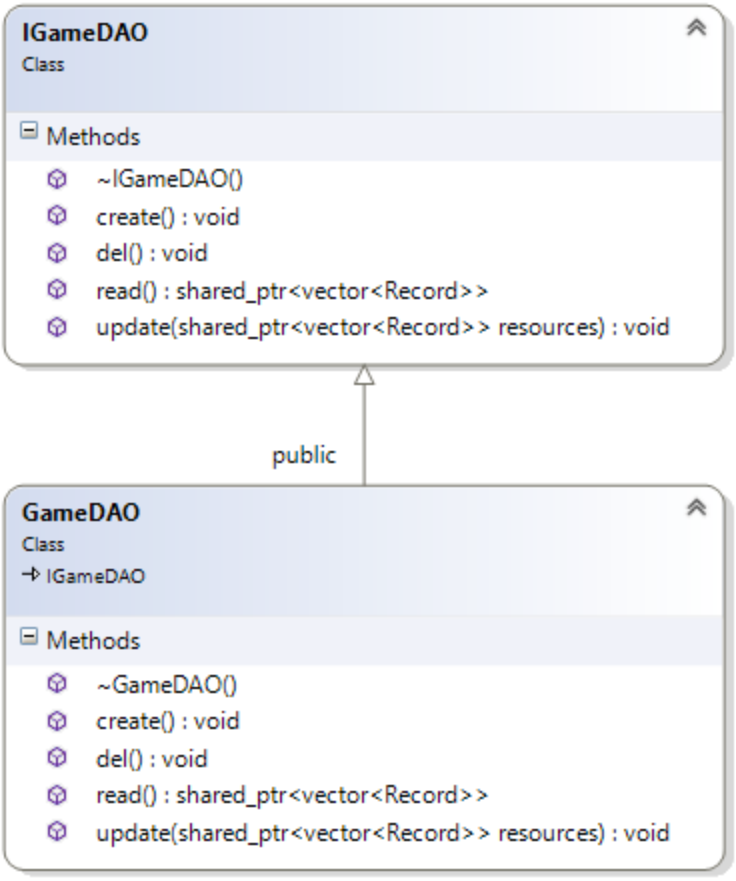

Figure 4.0: Game DAO class diagram

The DAO uses the TinyXML2[2] library to manipulate local XML files. To read the games XML config file the file is scanned and the contents are placed inside a vector of records. Each record contains a vector of resources. These resources can be identified by their names and searched using string matching. To get a particular resource the nested vectors are searched by name.

To update the XML file all game data is stored in a nested vector of Records and Resources and then passed to the DAO to be written to file using the TinyXML library. The Create function will create a new XML file and place the root element in the file, in this case the root element is identified by the name *"Config"*. The Delete function is not used, the XML file is overwritten with each update allowing records and resources to be removed beforehand.

5. Graphics

The graphics in the game are pixelated with minimal detail given to game object features. Bold primary and basic colours will be chosen for all game objects. The HUD labels and UI elements use the *"Gang Of Three"* font style.

- Characters
    - Player
    - Enemy Alien

The characters in the game need sprite images to be rendered. The player is running constantly and only needs one sprite sheet outlining 4 individual sprites for animations. The aliens are similar in that they need only one sprite sheet with 4 individual sprites for animations.

[2] TinyXML-2. 2015. TinyXML-2. [ONLINE] Available at:http://www.grinninglizard.com/tinyxml2/. [Accessed 14 April 2015].

Backgrounds

- Game Scene
    - Mountains background
    - Cave interior background

Backgrounds will be scrolled and a parallax effects will be implemented to create depth.

- UI Elements
- Menu Scene
    - Title Text
    - Start Game Button
- Heads Up Display
    - Pause Button
    - Distance and Gravity labels
    - FPS and Game Speed Labels
    - Booster Status and Lives Labels

6. Sound Effects and Music

No background music is used, there is only need for sound effects like foot steps, jumping, landing, and enemy movement. Sound is an extra feature of the game given enough time during development.

7. Schedule

The schedule[3] constructed for the games research, design, and development is contained in the spreadsheet shown in figure 4.0.

[3] https://github.com/damorton/AlienEscape/blob/master/Alien%20Escape%20Project.xlsx?raw=true

Alien Escape

David Morton

13.2.2015

Task	Est Time (hours)	Actual time (hours)	Complete	Notes
Get png image rendering to the screen	0.5	0.5	100%	issues with the SDL header files and libs folder
Add sprite class	0.5	1	100%	problem with pointers when moving the texture class out of the main.cpp file
Add world manager	0.25	1	100%	had to store a ref to the renderer for sprite class access
Add game manager	0.25	1.5	100%	difficult move game loop to the game manager from the main.cpp file
Add keyboard input	0.5	1	100%	
Get the player sprite moving	0.5	1	100%	issues with player in heriting the sprite class; player now holds info on sprite instance
Sprite animations	0.5	2	100%	implmenting player animations in the player class caused reference issues when trying to render the clipped areas of the sprite sheet
Add timer for delta time	0.5	0.5	100%	velocity value must be set high for player to move on screen
Move player by delta time	0.25	0.25	100%	limited player left and right movements and reset position to lower left corner
Add gravity to game world	1	2	100%	issues with gravity values
Add player jumping	1	1	100%	because of SDLs origin poistion it took longer than expected to figure out the velocity values
Add gravity flipping	0.5	0.5	100%	Added gravity direction value to the world manager for global access
Add renderer flipping for sprites	1	0.5	100%	Change renderer flipping in world manager when gravity flips
Add debug info	1	1	100%	
Add player score to debug	0.5	0.5	100%	
Background image	0.5	1	100%	
Scroll background images	0.5	0.5	80%	Issues with gap between the background images
Collision Manager	0.5	0.5	100%	Implemented collision detection into nodes and World Manager
State Management	0.5	1	100%	
Sprite frame coming from XML file	1	3	100%	
Menu scene configuration from XML	1	2	100%	
Game Scene config XML file	1	1	100%	
Character state management	1	2	100%	Check collision function needs pointer to other bounding box
UI elements from XML file	1	2	100%	String searching through records and resources, names needed for each element
Implement HUD	2	4	100%	Added scene class
Add pause functionality	1	3	100%	HUD needed to be outside of the pause block, delta time pause

Figure 4.0: Project schedule

8. *Future Work*

Given more time another enemy type could be added that would fly towards the player and add an extra challenge in trying to avoid it. Collectable items could also be added to provide the player with resources to purchase or use to aid them in the game like coins to buy weapons. Coins could also be used to customize the look of the player. Highscores would give the players a reason to replay the game, competing with their best or with a friends best score. Competition with others creates valuable replayability value.