Technical Design Document

The Second to Last of Us

Adam Frewen – 0813664
Andrew Pike – 1299530
Seong Jho – 13829220

Contents

Target Platform

- PC Windows OS

Development Environment and Tools

- Visual Studio 2013 - Main development IDE
- GPFramework - main structure of game
- TortoiseSVN - version control and file sharing

Required Technology

- XBox controller (360 and above)

Key Technical Challenges and Possible Solutions

Zombie swarming

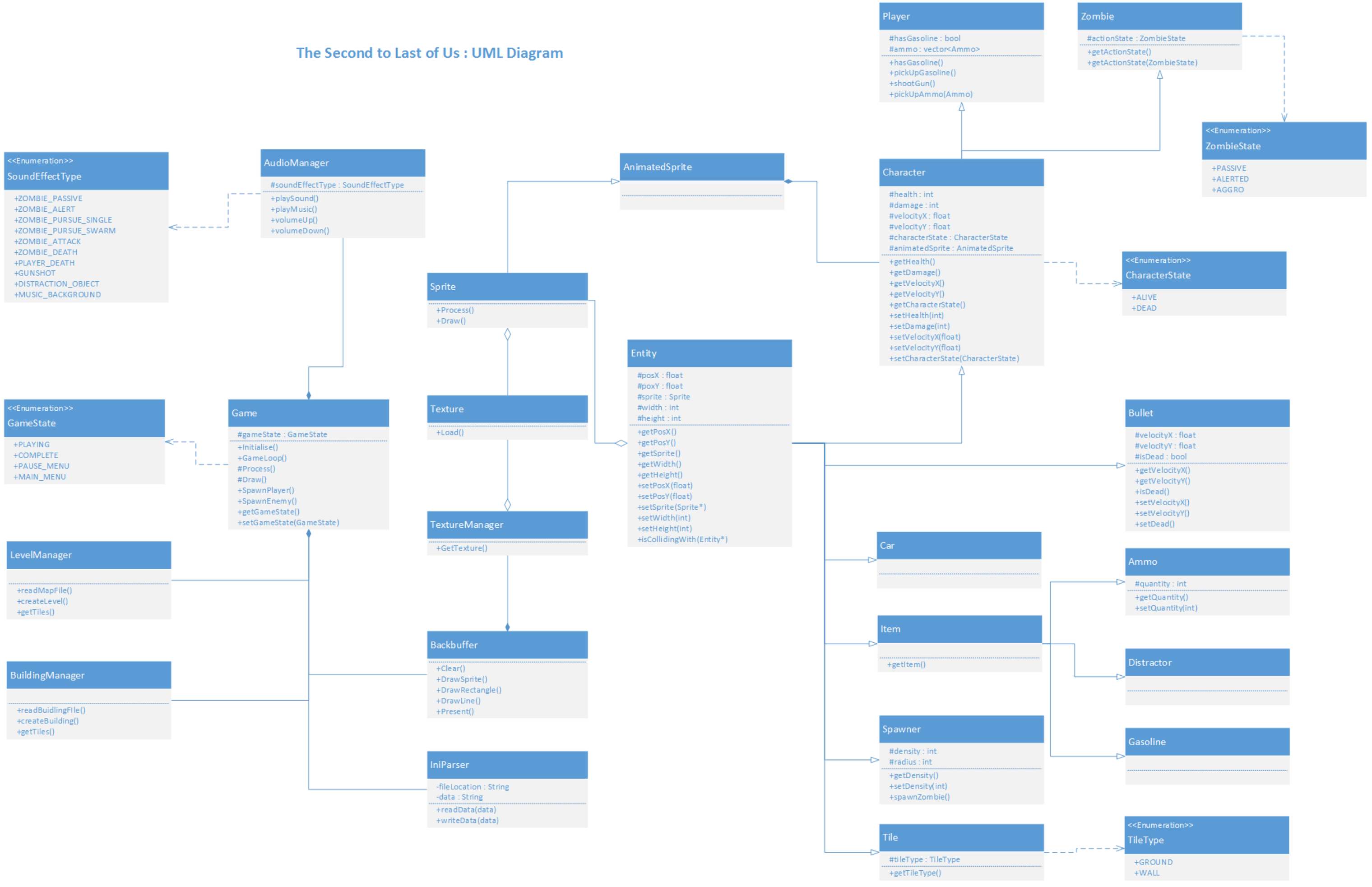
- This type of AI has not been dealt with before, therefore implementing the zombie to move in a swarm/flock will be a challenge to us.
- We have set aside time in our schedule so that if the AI has caused us problems which delay the development, we will be able to get the whole team to focus on the AI.

Development Methodology

We are attempting to follow an agile methodology with team stand-ups on Tuesdays and Fridays, and a weekly milestone one the Saturday of the week.

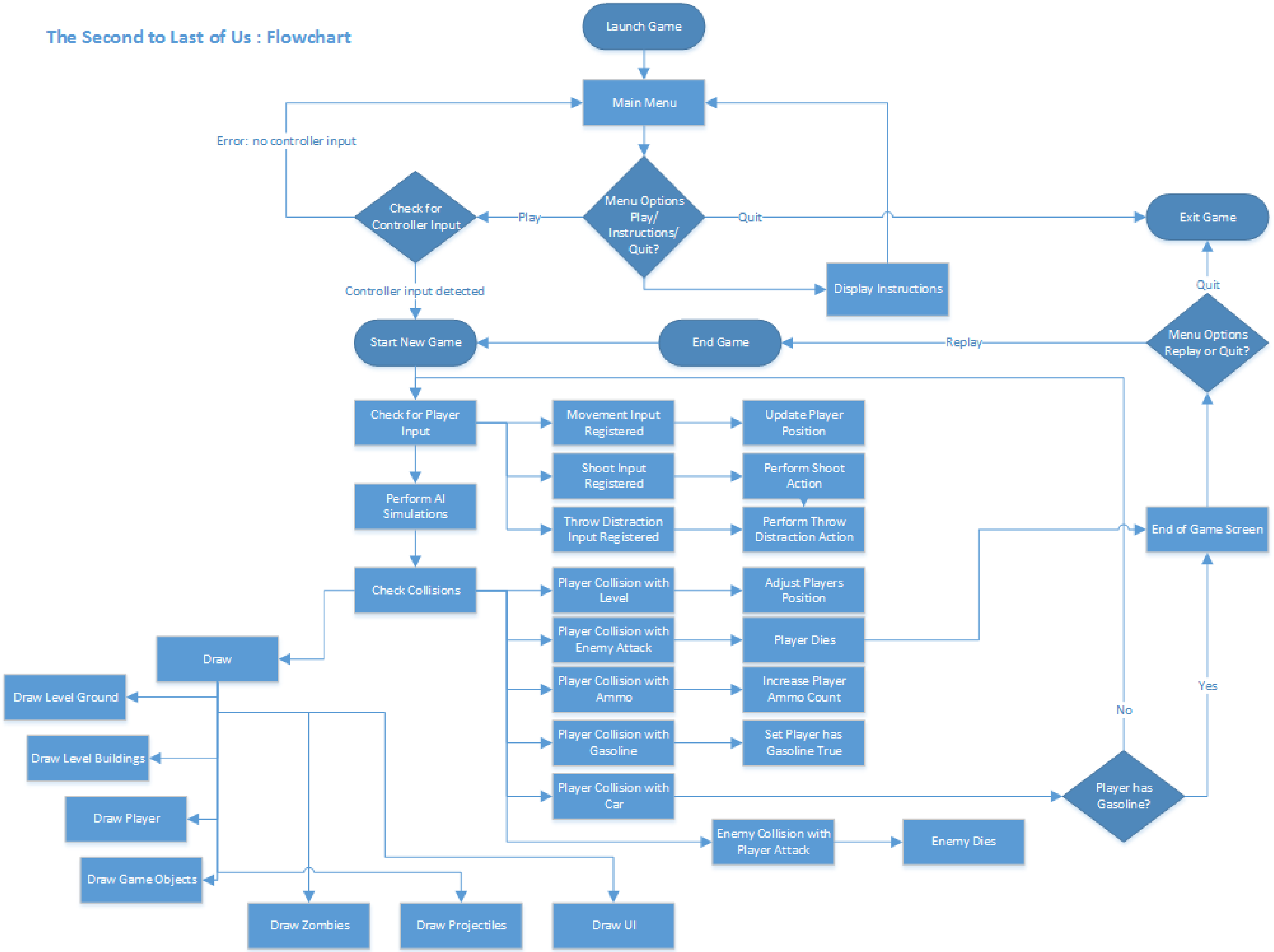
Architecture Overview

UML Diagram



The Second to Last of Us : UML Diagram

Game Flowchart


The Second to Last of Us : Flowchart
Launch Game
Main Menu
Error: no controller input
Check for Controller Input
Play
Menu Options Play/ Instructions/ Quit?
Quit
Exit Game
Display Instructions
Controller input detected
Quit
Start New Game
End Game
Replay
Menu Options Replay or Quit?
Check for Player Input
Movement Input Registered
Update Player Position
Shoot Input Registered
Perform Shoot Action
Perform AI Simulations
Throw Distraction Input Registered
Perform Throw Distraction Action
End of Game Screen
Check Collisions
Player Collision with Level
Adjust Players Position
Draw
Player Collision with Enemy Attack
Player Dies
Draw Level Ground
Player Collision with Ammo
Increase Player Ammo Count
Yes
No
Draw Level Buildings
Player Collision with Gasoline
Set Player has Gasoline True
Draw Player
Player Collision with Car
Player has Gasoline?
Enemy Collision with Player Attack
Enemy Dies
Draw Game Objects
Draw Zombies
Draw Projectiles
Draw UI

Key Algorithms

Key Algorithms which govern gameplay elements.

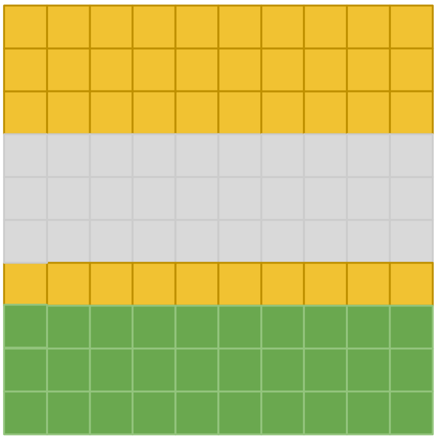
- Painters Algorithm
    - Assists with sorting 2D sprites depth order based on priority.
    - Priority order (Highest to lowest)
        - Player
        - Zombie
        - Bullet
        - Distraction throwable
        - Car
        - Building
        - Ground
- Tiling Algorithm
    - The level layout will be generated based on a .txt input file utilising single character symbols to identify the required tile type for each portion of a grid. These tiles will be stored in a vector. Tile types will be handled using an enumerated type. This is detailed further in Map Generation.
- Collision
    - Box2D will be utilised to handle collisions.
        - Player and Zombie collisions will be resolved using a circle body.
        - Walls and game object collisions will be resolved using a square body.
        - Bullets and distraction projectiles utilise Box2D collision. These assets will be reused as the player will have a limited supply.
    - Circle to circle collisions will be utilised to calculate the places collision with the car and gasoline game objects

Map Generation

The map will be created by parsing a text file that details the map, different letters will correspond to different tiles being created in the game. A level will consist of several map tiles connected into one level.

```
DDDDDDDDDD
DDDDDDDDDD
DDDDDDDDDD
RRRRRRRRRR
RRRRRRRRRR
RRRRRRRRRR
DDDDDDDDDD
GGGGGGGGGG
GGGGGGGGGG
GGGGGGGGGG
GGGGGGGGGG
```

D = Dirt
R = Road
G = Grass

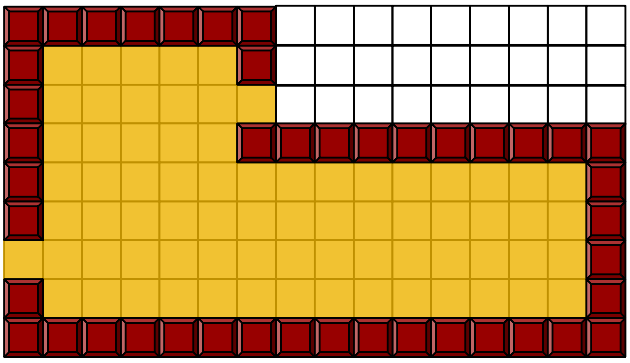


Example Map piece

Then the map will be overlayed with buildings which will be created similarly to the map by parsing text files with set building layouts.

```
WWWWWWWBBBBBBBBB
WFFFFFWBBBBBBBBB
WFFFFFFBBBBBBBBB
WFFFFFWWWWWWWWWW
WFFFFFFFFFFFFFFW
WFFFFFFFFFFFFFFW
FFFFFFFFFFFFFFFW
WFFFFFFFFFFFFFFW
WWWWWWWWWWWWWWWW

W = Wall
F = Floor
B = Blank
```



Example Building

Entity Movement and Collision

Box2D will be used to resolve majority of the collisions required for gameplay. Most of which are detailed in their relevant sections.

Player movement

Movement will be handled through Box2D by applying a force to the player based on the input value of the left analog stick. The player's sprite will be rotated according to a facing variable which is controlled by the right stick.

Player interaction

Pressing the fire button (RT) will cause a bullet entity to be propelled from the player in the direction the player is facing. The bullet will keep going until it goes past the player's gun range attribute or hits another entity. If it hits a zombie, the zombie will die.

Pressing the distraction button (RB) will cause a distraction entity to be propelled from the player in the direction the player is facing. It will move for a fixed distance then stop, or stop if it collides with another entity. After stopping the distraction entity will then raycast a fixed distance, and switch any zombies that are not in an agro state to the distraction state.

Once the player finds the gasoline can, they can move into it to collect it. This causes the gasoline can entity to be destroyed and the Player's has Gasoline variable is set to true. The player can then move to the car to interact with it. If the player has the gas, the game ends.

Zombie Spawning

A number of spawn points will be randomly created around the map.
Spawn points have given radius within which zombies can be spawned.
Number of spawn points can be specified for individual levels.
Number of zombies spawned at a spawn point can be specified and zombies will be spawned at random locations within the radius of the given spawn point.
Spawning collision is checked after spawning zombies. Contact list for each zombie is checked. If colliding with a wall. Move the zombie out of the wall by applying a force in a random direction recursively until no collision is detected.

Artificial Intelligence

AI techniques will be implemented with reference from Mat Buckland’s “Programming Game AI by Example” (Buckland, 2005).

Zombie Behaviour

Zombies are only processed if they are on screen or not in passive state.
All processing zombies will use Box2D to ray-cast a fixed distance to try and determine player’s location.
Zombie AI will be based on finite state machine. Each zombie will have an enum storing the current state of that zombie.

States

- Passive: Wanders aimlessly until Player is detected within the alert radius.
- Alert: Same behaviour as passive, but if the Zombie remains in this state for too long, or the Player moves too close to the Zombie, it will transition to Agro state.
- Agro: In the Agro state, the Zombie will move towards the last detected location of the Player. If the Zombie reaches the Player, it will attack. If the Zombie gets to the last known player location but cannot detect the Player, it moves back to Passive.
- Distracted: If the Player’s distraction detects the Zombie, it will move to the location of the distraction then stay distracted for a fixed amount of time before returning to Passive state. While in the distracted state the Zombie will not detect the Player.

Movement & Pathing

The movement speed of the zombies will be set to 80% of the movement speed of the player.
We will be using the Seek, Wander and Object & Wall avoidance algorithms explained by (Buckland, 2005) to handle zombie movement. While zombies are intended to chase the player, we think simple direct seeking behaviour fits a zombie’s expected low level of intelligence as opposed to something more advanced like a pursuit behaviour.

Horde behaviour:

Implement flocking algorithm detailed by (Buckland, 2005) when three or more zombies are in close proximity. This will allow zombies to move as a pack.

Debug Features

- Frames per second display
    - FPS display to help track FPS drops during.
- Reduce game speed
    - Manual reduce the tick speed so gameplay can be observed in slow motion.
- Game pause
    - Live pause (without UI clutter) to allow aspects of the game to be inspected while stationary.
- Collision display
    - Visual display of collision. Collision boxes should change colour when collision is detected.
- Draw last sighted object
    - Draws a placeholder sprite at the position where the player was last sighted by zombies.

Testing Plan

- Game over condition successful?
    - Game lost when player health is reduced to zero.
- Victory condition successful?
    - Game won when player reaches car with gas.
- Is collision performing as expected against all types?
    - Level, Player, Zombie, Bullet
- Are any memory leaks present?
- Is game performance optimized?
- Does ammo pickup provide correct increments?
- Are all assets loading correctly?
- Does all player abilities function as intended?
    - Movement
    - Turning
    - Firing the gun
    - Throwing distraction
- Does picking up the gasoline can correctly update the status?
- Do Zombies behave correctly?
    - Spawn correctly
    - Alert/Agro/Distract under right conditions
    - Path to point correctly
    - Object/Wall avoidance
    - Flocking/Horde functioning

Risks

We have not attempted any game AI this advanced before.
Only one of our team members has experience with Box2D.

Naming Schemes

Camel casing will be utilised for all assets.

- Classes: ClassName
- Variables: mf_variableName (member float)
- Enumeration: ENUM
- PNG: SpriteImage
- Audio: SoundFile

File Formats

- .png
  - PNG format will be used for all sprite sheets and 2D assets.
- .mp3 & .wav
  - MP3 and WAV format will be used for all game sound effects and music.

Third-party APIs and Middleware

- Box2D
- FMOD

References

Buckland, Mat. (2005). *Programming Game AI by Example*. Plano, Texas: Wordware Publishing.

Team Signoff

Adam Frewen	AFrewen
Andrew Pike	Andrew Pike
Seong Jho	Seong