**Guardian of the Outlands**

****

**Designed by Tom Roberts**

**15th March 2015**

**For PC (web)**

**Ages: 12+**

**Finish date: 28th May 2015**

**Detailed Design**

**Game Outline**

The player will spawn at a random location on the screen. Each enemy will spawn at a location on the edge of the screen that is chosen randomly at the start of each wave.

Difficulty will vary randomly over successive waves by spawning random numbers and random types of enemies, such as quick-moving jeeps that can pick up a barrel and escape quickly, and slow, tough tanks that take more bullets to kill.

Players score points by killing an enemy (each type of enemy gives a different score) and earn points for each item remaining on the screen at the end of a wave. If they crash into an enemy, they will lose points and be respawned at a random position. The game ends when all of the barrels have been stolen. The player’s final result is their cumulative total of points, the number of waves they survived, and the time taken.

The player can collect power-ups that give bonuses including: increased damage, invulnerability, double points. Obstacles may also appear that will hinder the player, such as oil spills that change the player’s direction randomly, patches of tar that slow movement etc.

**Software**

The game will be created using Unity as it is often used to develop games for websites (played via the Unity Web Player browser plugin), but it also provides ways to easily deploy to other platforms if needed. It is one of the most established game creation systems[[1]](#endnote-1) and therefore has a plethora of documentation, and official and community-contributed tutorials[[2]](#endnote-2).

Unity also offers many libraries and functionality that the game will need (basic physics, collision detection).

**Hardware**

The game will be developed for the PC as this would be the main platform where people play games on the web. The hardware requirements of the game will be kept minimal and will not need high-end components to run, leaving the possibility of deploying to mobile platforms in the future open.

Targeting the PC also allows the use of the keyboard and mouse as game controls for the player to move their tank and fire its weapon, something which may be more difficult to do using touch/accelerometer mobile controls.

**Graphics**

2D sprites will be used for all of the art as it is a top-down 2D game, so does not require 3D models. Vector graphics could have been used but sprites give satisfactory detail for the game. There will be sprites for the player, enemies, barrels, pick-ups, hazards, the menu system, bullets, explosions etc. All of the art will be sourced from websites that offer art that are in the public domain (so have no limitations on use), or have been released under a Creative Commons license. Credit to the creators will be given in a README text file or credits section of the game in the manner that the license stipulates.

Examples of the graphics intended to be used can be seen in Appendix 1.

**Audio**

**Music**

The main menu of the game will have looping background music sourced from websites that offer Creative Commons and Public Domain-licensed music. The track chosen will need to be upbeat enough to convey the action style of the game but considering that it could be looped many times if a user is browsing the menu, it should not be too overbearing.

During gameplay, there will also be looping background music that is upbeat and will be used to create tension and action whilst playing.

**Sound effects**

There will be no sound effects in the main menu as I feel that sounds played when hovering or clicking a button, or transitioning between menu screens, would become quite annoying after a while; feedback on these events are instead given visually.

During gameplay, there will be sound effects for the player firing their weapon, enemies exploding, and a barrel being stolen off the edge of the screen (accompanied by visual feedback). An engine sound effect for the player and enemies moving may be used, if it is not too annoying to have the same effect being triggered frequently.

**Code Structure**

Two design patterns will be used: the Factory design pattern and the Object Pooling design pattern.

**Factory Design Pattern**

This pattern will control how and when elements of the game are spawned. There will be spawners for the player, enemies, pickups etc that will be responsible for monitoring the game states and instantiating objects, as needed.

**Object Pooling Pattern**

This pattern is best suited for games that require a lot of instantiating and destroying objects, or repeating some other expensive memory operation. As this is a shooter game, object pooling will be used for any firing of bullets. Any game entity that needs to use bullets (such as the player) will check objects in and out of a pool rather than repeatedly doing costly memory allocations to create/destroy bullet objects.

**MVC**

To adhere to MVC, an object’s animation and movement (view) will be kept separate from the variables that represent its internal state (model). This will be done by splitting object controllers into several scripts that reference each other i.e. an enemy colliding with a barrel will be detected in the view, which will trigger a change in its state in the model.

**Class diagram (how classes/scripts will fit together)**

**Activity diagram showing the behaviour of the code**

**Game-Specific Features**

**Collision Detection**

Collision detection is the basis of most of the gameplay. The player is able to shoot bullets that collide with enemies to destroy them. If the player collides with an enemy, the player will be destroyed and respawned at a random position on the screen. Collision detection is also used to implement an enemy picking up a barrel by bumping into it.

Around the edge of the game screen, there is an invisible collider that will destroy any objects (except the player) that exits the screen. This is to prevent memory leaks caused by objects moving out of the field of view but not being destroyed (i.e. stray bullets).

**Artificial Intelligence**

Artificial intelligence controls the enemies’ actions. They have three modes: No Target, Move, and Escape.

The ‘No Target’ state means that there are no free barrels for the enemy to move towards and try to pick up. This usually happens when there are more enemies than barrels on screen due to some having already been stolen. The enemy will roam randomly around the screen whilst repeatedly scanning the game scene for a barrel that no other enemy is already moving towards. The enemies are intelligent enough to know that if they accidentally bump into a barrel whilst roaming, they will pick it up and switch to ‘Escape’ mode.

The ‘Move’ state is used for when the enemy has a barrel targeted and can move towards it. The enemies are intelligent enough to know that if they accidentally bump into a barrel that wasn’t their intended target, they will abandon their original target and pick up this new barrel and switch to ‘Escape’ mode.

The ‘Escape’ state means that the enemy has picked up a barrel and must try to escape to its randomly designated exit point somewhere on the edge of the screen. An enemy can only carry one barrel at a time so if it bumps into any other barrel whilst trying to escape, it will simply ignore it.

**Physics Engine**

The physics engine is used to move bullets across the screen at an arbitrary speed.

**Time Management Plan**

**Tasks implemented during prototyping/early development:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Monday 16th March** | **Tuesday 17th March** | **Wednesday 18th March** | **Thursday 19th March** | **Friday 20th March** |
| **Task** | Source sprites for: player, enemies, bullets, explosion | Factory design pattern (spawn player, enemies, barrels) | Player movement | Player shooting and destroy enemies | Preliminary enemy AI – Move mode (picking up barrels) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Monday 23rd March** | **Tuesday 24th March** | **Wednesday 25th March** | **Thursday 26th March** | **Friday 27th March** |
| **Task** | Preliminary enemy AI – No Target mode | Preliminary enemy AI – Escape mode | Add new type of enemy - Jeep | Object Pooling design pattern for bullets | Preliminary scoring system |

|  |  |  |
| --- | --- | --- |
|  | **Monday 30th March** | **Tuesday 31st March** |
| **Task** | Main menu system | Main menu system |

**Tasks to be completed:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Monday 11th May** | **Tuesday 12th May** | **Wednesday 13th May** | **Thursday 14th May** | **Friday 15th May** |
| **Task** | Refactoring code for MVC | End game | Pickups | Pickups | Hazards |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Monday 18th May** | **Tuesday 19th May** | **Wednesday 20th May** | **Thursday 21st May** | **Friday 22nd May** |
| **Task** | Enemy hit points | Add new type of enemy – Heavy tank | Change enemy ‘No Target’ mode to Pac-Man-style player chasing | Change enemy ‘No Target’ mode to Pac-Man-style player chasing | Audio (music & sound effects) |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Monday 25th May** | **Tuesday 26th May** | **Wednesday 27th May (day before deadline)** | **Thursday 28th May (deadline)** |
| **Task** | High scores | Gameplay testing/tweaks | Gameplay testing/tweaks | Verify final build runs + submit |

**Appendices**

**Appendix 1**

|  |  |
| --- | --- |
| **Player** | **Enemies** |
|  |  |

|  |  |
| --- | --- |
| **Bullets** | **Explosions** |
|  |  |

|  |  |
| --- | --- |
| **Barrel** | **Power-ups** |
|  | (set to different colours to indicate which power-up it is) |

|  |  |
| --- | --- |
| **Hazards** | **Menus** |
|  |  |

**References**

1. Unity Technologies. (2015). Public Relations. Available: http://unity3d.com/public-relations. Last accessed 15th Mar 2015. [↑](#endnote-ref-1)
2. Unity Technologies. (2015). Learn With Unity. Available: http://unity3d.com/learn. Last accessed 15th Mar 2015. [↑](#endnote-ref-2)