WINDS OF DEATH
GAME DESIGN DOCUMENT

Aleksandar Marinov (B00351200), Mohammed Latif (B00333837), Fraser McAulay (B00371477), Mihai Maftei (B00359190)

Abstract

Winds of death is a top-down shooter style game, available on our website. The core audience for this game would be young adults and teenagers

Contents

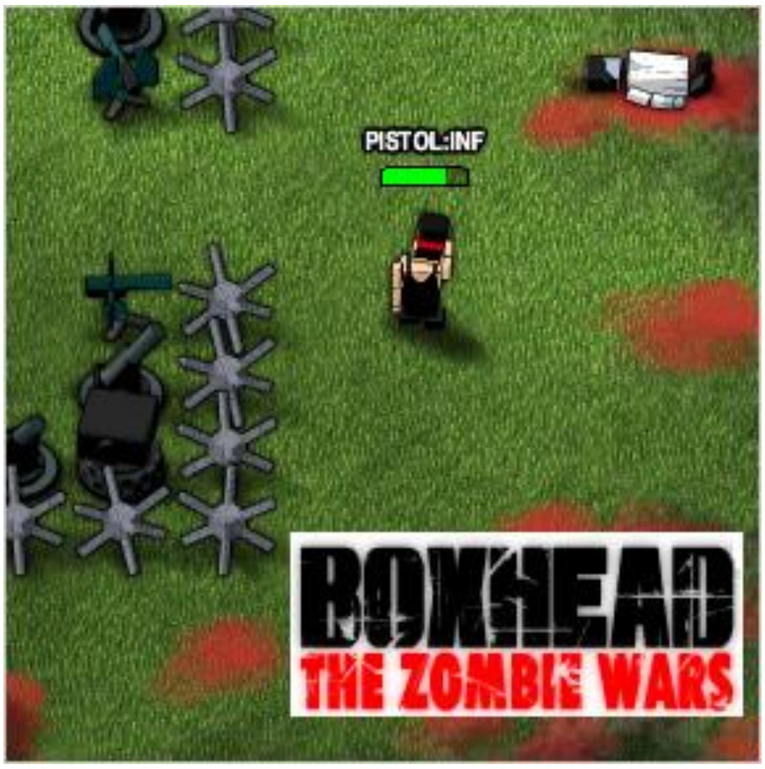
Game Design

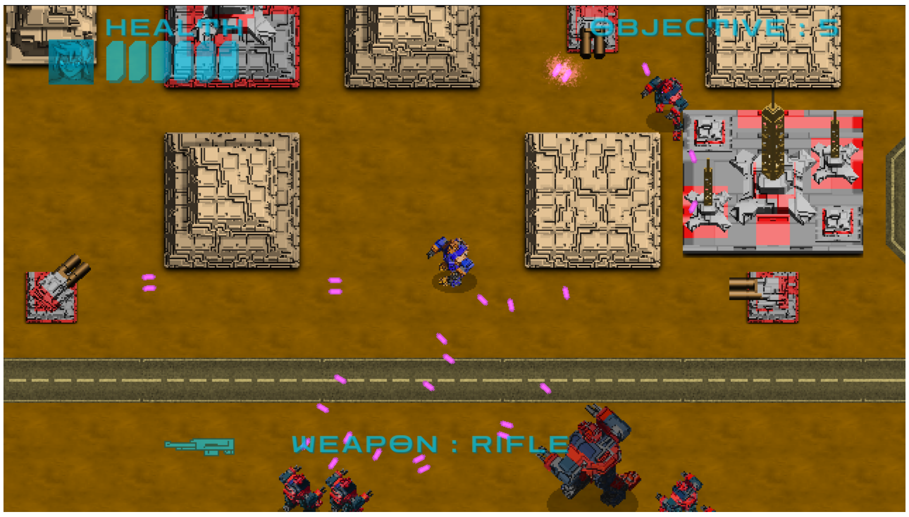
Game Overview

Winds of death is a top-down shooter style game, available on our website. The core audience for this game would be young adults and teenagers. We chose this audience as they are the ones most seen to play shooter games. To make this game, we will be using WebStorm from JetBrains and Phaser Editor 1.5.4 as a development environment. The game will be made with the framework Phaser version 3.55.2 "Ichika", and the main programming language will be JavaScript. The goal of the game is to kill as many zombies as possible while progressing through a series of linear levels. There will be opportunities to use different guns throughout the levels to aid player progress.

Game Concept

We chose to design *Winds of death* as a 2D co-op top-down zombie shooter, a subgenre that we think is extremely attractive for our desired audience. This idea came from one of our developers. He mentioned the style of game and we started talking about it and looking up past browser games with a similar style. The games we got ideas from where: *Boxhead: The Zombie Wars*, *Zombies Ate My Pizza*, *Dead Estate* and *Mecha Team Maverick*.



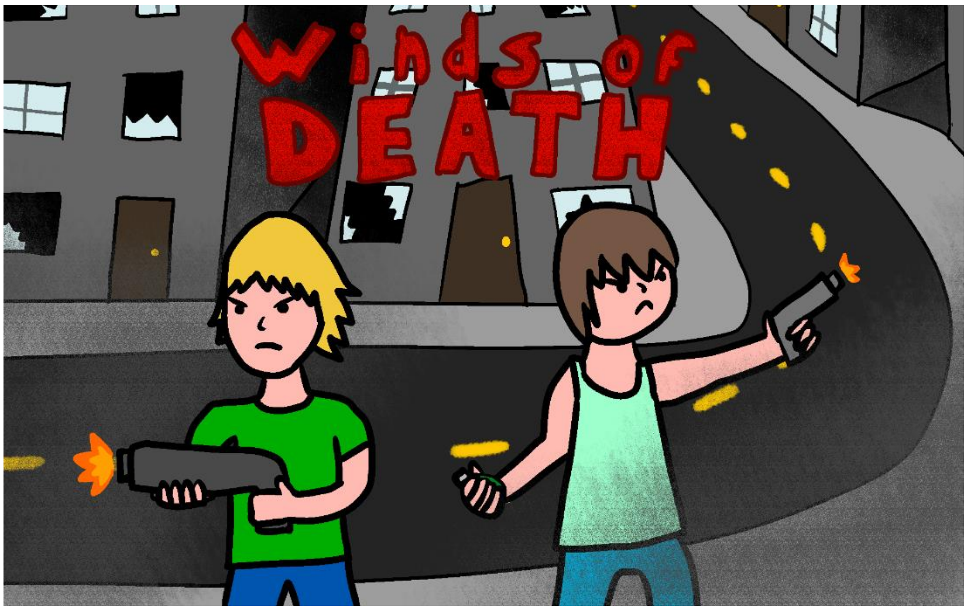


These games all utilise 2D top-down shooter gameplay in order to create challenging, addictive and endlessly replayable gameplay loops. We seek to create a similar gameplay experience in our game: *Winds of Death*. In our own game, the protagonists progress through levels by shooting as many zombies as possible while dodging their attacks. The two players must rely primarily on reaction times and intuition to survive. Similar to the games we were inspired by the players will be able to get weapon upgrades and increase their score by killing enemies.

Characters

In this game. Our characters Hannibal Lector and Norman Bates are survivors in a world where a zombie apocalypse has recently set in. Starting out in the city of New Glasburgh, they must fight their way through various locations including a forest and a river in order to find a boat and escape the mainland. Unfortunately, they're vastly outnumbered by the zombies so they'll need to scrounge up whatever firepower they can to succeed on their journey.

Plot



In *Winds of Death*, you and your co-op partner play as a pair of survivors in a world where a zombie apocalypse has recently set in. Starting out in the city of New Glasburgh, they must fight their way through various locations including a forest and a river in order to find a boat and escape the mainland. Unfortunately, they're vastly outnumbered by the zombies so they'll need to scrounge up whatever firepower they can to succeed on their journey

Gameplay

The core gameplay loop revolves around shooting mechanics working in conjunction with a scoring system. The player will need to kill many zombies to progress to the next level, the final of which will be much larger and full of enemies thus serving as a test of the players skills. Since the game will be a 2-player co-operative experience, both will have their own scoreboard. This will serve to encourage competitive play and increase replayability.

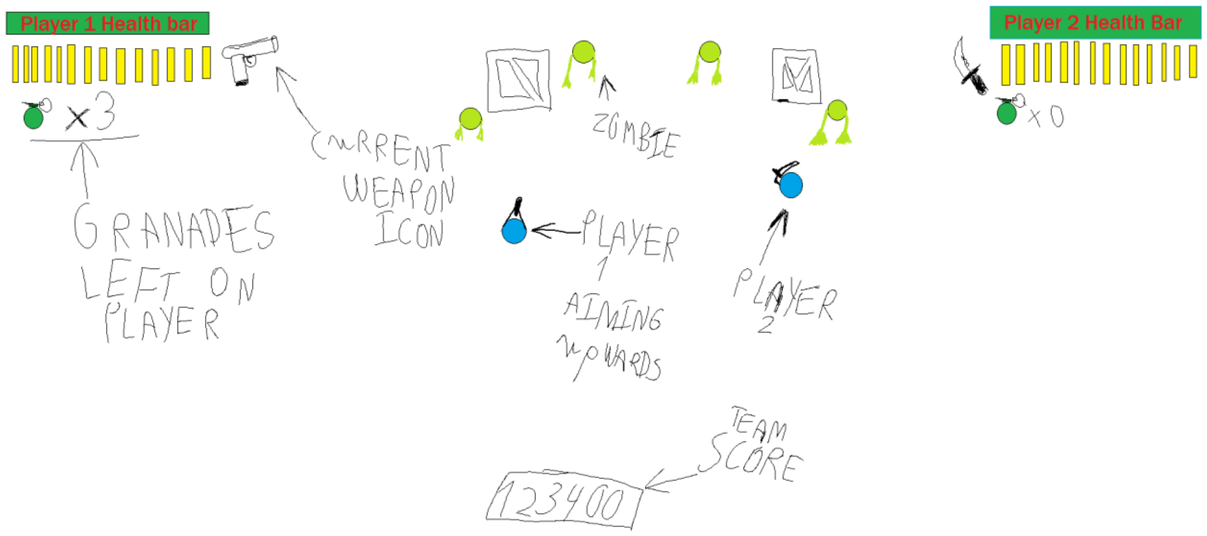
Both players will start each level with a pistol but will be able to pick up new weapons such as shotguns and assault rifles as they progress. Each new weapon will be available on the ground as a pick-up object and will have a different fire mode and rate of fire, providing unique advantages and disadvantages to each. This will also help enhance the competitive nature of the game as there will be a limited number of pick-ups, meaning that the player to reach it first will gain an advantage. There will also be grenades as one-use throwable items.

In the game, the zombies will come in a few different varieties. These will include fast, slow, and regular zombies, each having several hit points inversely proportional to their speed. Each enemy in a particular level will spawn semi-randomly, having a set point to appear from with slight randomness applied to positioning when spawned. Enemy type will be randomised too in later levels to keep the players alert and ready for unexpected challenges.

Game Interface Design

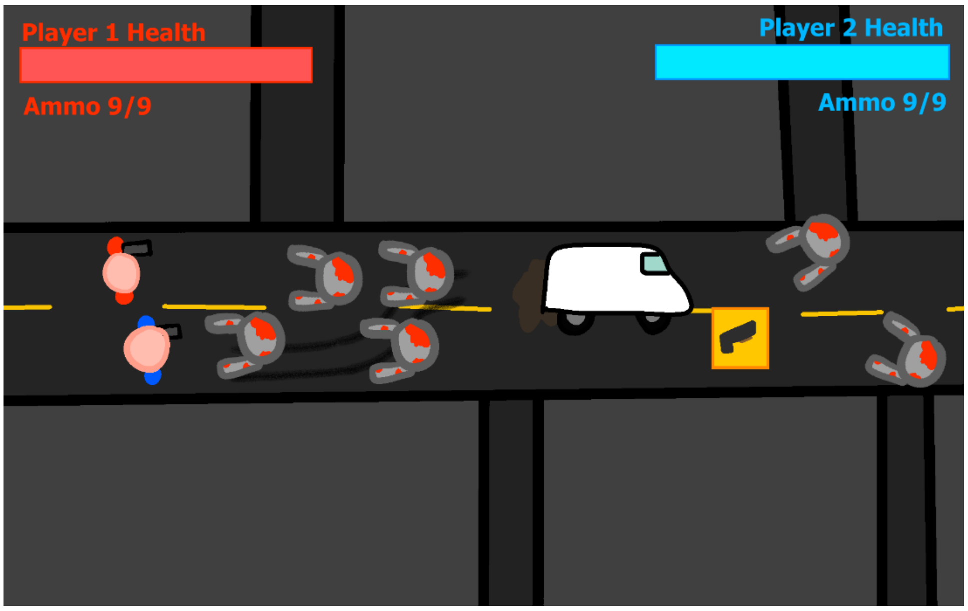
The user interface of our game will be in keeping with the theme of our game, making use of gritty visual elements and blood motifs throughout. The main menu will consist of a button to begin the game and another to view the game credits. The design of the main menu will inform the player of the type of environments they can expect to see in the game's levels.

The in-game UI will be as streamlined as possible, only showing information the player needs to know. These elements will include player health bars, score counters, number of lives, current weapon indicators and current objectives. Elements unique to the first and second player (health, score, weapon) will be grouped in the top left and right corners of the screen respectively. Each visual element will adhere to the game's apocalyptic motif.

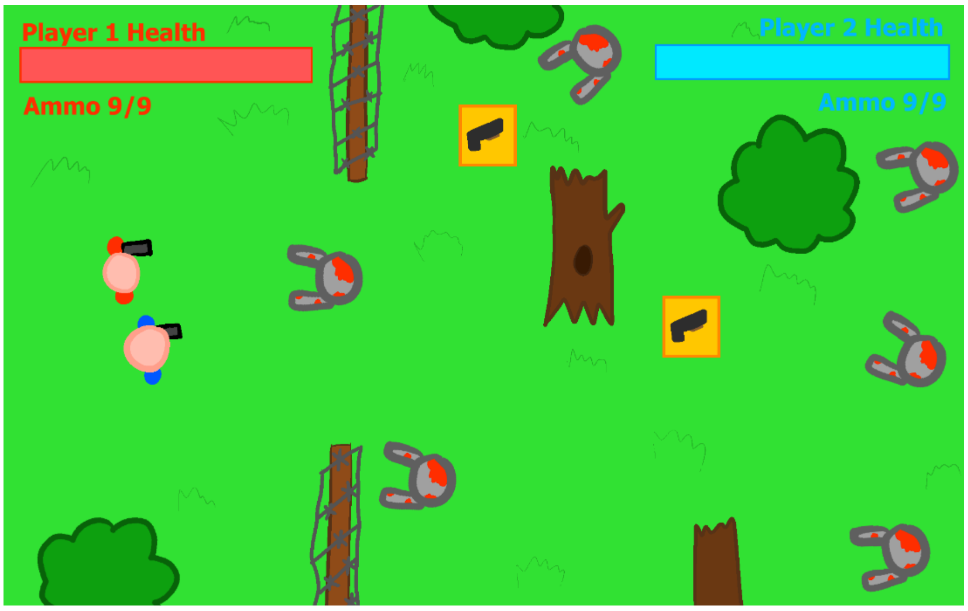


Game Environment/Level Design

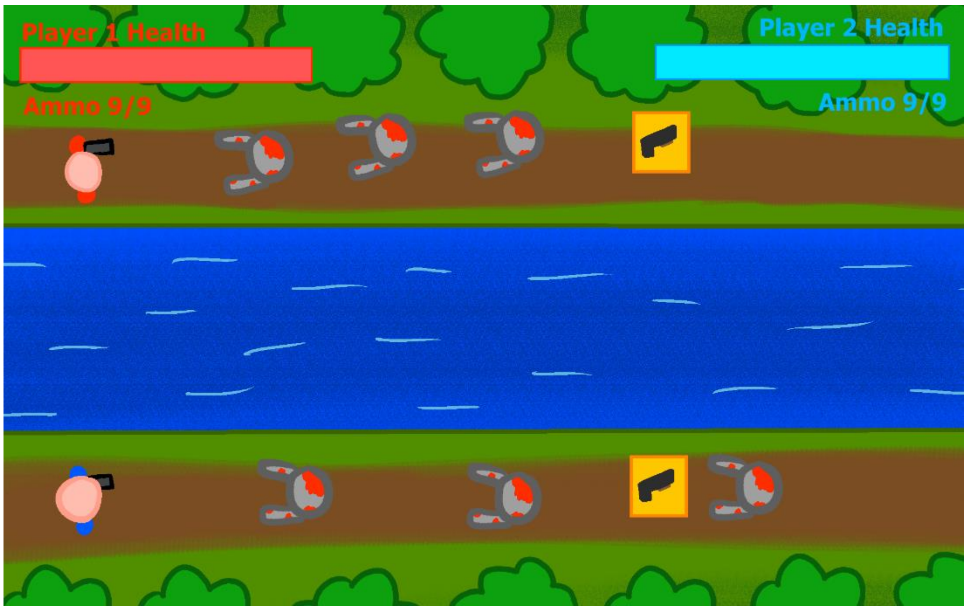
Winds of Death will feature several unique environments across its multiple levels. Each of these areas will offer different challenges through their layout and enemy distribution. Firstly, there's the city area which serves as the introductory level in the game. The level will feature long straight streets with buildings and occasional alleys for shortcuts and enemy spawns. Cover will be available in the form of overturned cars and barricades from before the city was overrun. The enemy distribution of this level will mostly be medium-sized groups of slow zombies so as not to immediately overwhelm players.



The second level will be an open field with a few trees here and there. This will serve as the introduction of faster zombies with the more open space – compared to the city – giving the player more time to react to this new enemy type. Fallen trees will also be placed sporadically throughout the level to serve as cover for players. There will also be barbed wire fences which serve as both cover and a hazard that will deal damage to anything it touches. Players will have less trouble with this level once they realise that they can lure zombies into the hazards, allowing for more creative kills and conserving ammo in the process.



The third level will take place along the banks of a river, starting with either player stuck on either side. They will be required to fight their way along the river separately for a short while before reuniting at a dock and heading to a boat, this being their final objective. The main challenge of this level will be the initial separation along with the lack of cover.



We believe that these levels will each provide their own unique challenges with their unique layouts and characteristics. That said, we do aim to add more levels in future developments as there are still plenty of possibilities left to explore.

Game Implementation

Game Features and Future Development Plans

Winds of Death will use the Standard control scheme for a 2D top-down shooter. These controls will include moving with the W, A, S, D keys for Player 1 and using the Arrow keys for Player 2. Other games might use the mouse to assist with aiming but considering our choice to use local co-op, it is best to keep the aim direction to the movement keys. To shoot, the player presses either Space or the Right arow key. We aim to make the gameplay revolve around a competitive scoring system to encourage playing more aggressively, incentivising both players to outperform their partner.

In the future versions of the game, we would like to add the following:

Attacking abilities such as increasing and upgrading the damage that the players weapons cause and increasing and decreasing the firing rate of their weapons. In addition to implementing a weapon pickup system.

Defensive abilities such as upgrading the resistance of player shields and implementing a health pickup system.

Future additions may include the addition of more game levels. There would also be a choice in the options scene for setting the level of difficulty for the game.

Gameplay & Player Mechanics

The central mechanic of the game will be its shooting mechanic. The player will be able to enhance their characters' shooting capabilities by collecting pickups for different weapons such as a shotgun or assault rifle, with their starter weapons being pistols. As an extension of the shooting mechanic, one-use grenade pickups will also be available to use. These will be a useful tool for players to clear space in a tense situation where perhaps they are surrounded by too many enemies, unable to shoot enough to make an opening in time to escape. To accentuate their importance as an emergency measure, players will only be able to carry up to 3 grenades at a time. When the player shoots a zombie, they get ten points added onto their score. Stronger guns will have a 2x multiplier while grenade kills give 3 times as many points per kill.

Pseudocode

Grenade function

```
Key pressed "Q"
    {
        If (grenades > 0)
        {
            grenades -= 1
            create grenade(at player position, using player rotation)
        }
        Else
        {
            print("No grenades left!")
        }
    }
```

Player Code

```
Switch (key pressed)
    {
        W pressed:
            Move player up
        A pressed:
            Move player left
        D pressed:
            Move player right
        S pressed:
            Move player down
        Q pressed:
            Throw grenade
        E pressed:
            Shoot gun
    }
```

Design Pattern

For our game's development we hope to utilise the Singleton design pattern. We decided that this would be an effective way to approach development as it encompasses using a central class for any particular element that can be accessed globally as it only one instance is able to be instantiated. This would be useful as it would simplify creation of characters in our game and avoid unnecessary confusion in the process.

Project Management

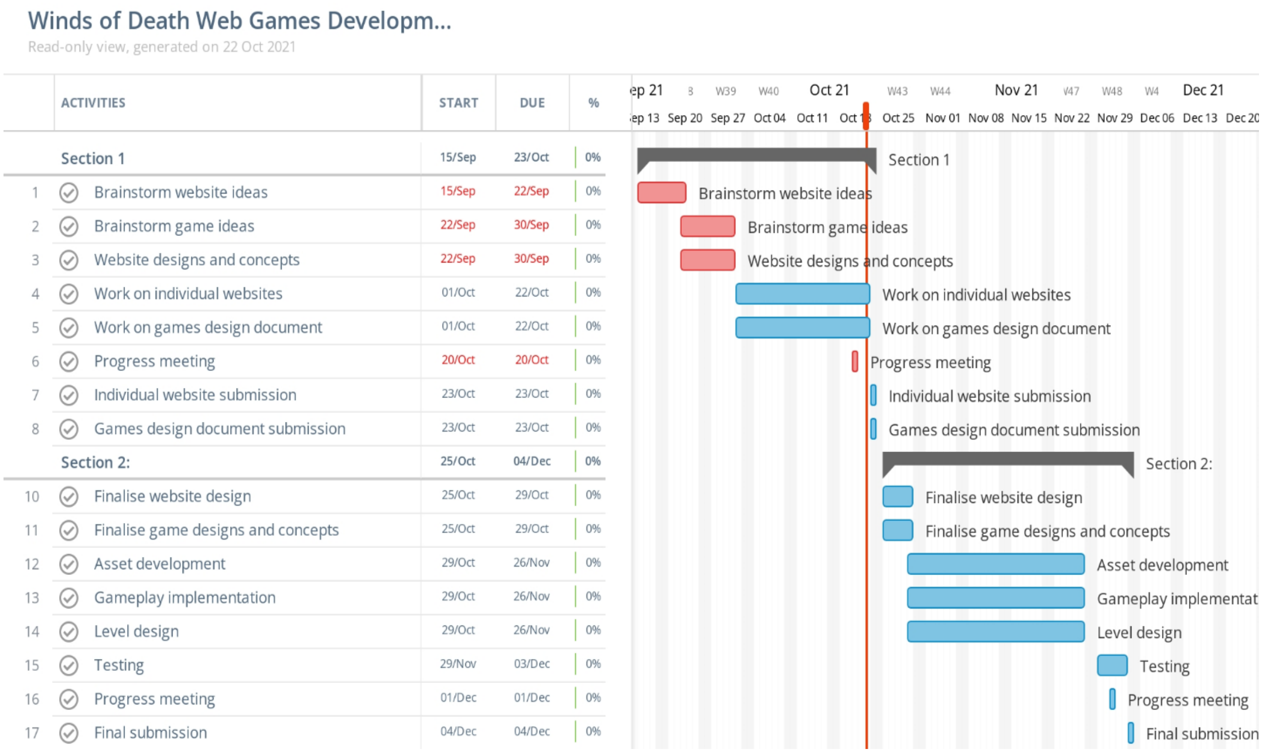
We decided to choose the Agile development methodology for the web games development two project. This enabled us to use more of a flexible creation process, as it allowed us to use more a frequent iteration of the design and development procedure. Therefore, this mean that the project design and models were developed together, with frequent meetings throughout we established timelines and specifications for our project aspirations.

This meant that as team members we all had to communicate together effectively and efficiently throughout the project development. To help us communicate with one and another we used Microsoft Teams and Discord to allow us to voice chat, text messaging and screen share for the purposefulness of development and implementation meetings.

We will be utilising Microsoft OneDrive as the repository for this project to manage and view game files, developer contributions and production history. The Microsoft OneDrive page is linked here: https://bit.ly/3EKy8bp.

In addition, we have also made a project plan and a Gantt chart that we used to set milestones, dependencies, assign issues, risks, and tasks for the project requirements. It has enabled us as a team to better understand our working responsibilities and how our efforts have an impact on the overall project. The overall project plan, Gantt chart and UML diagram is included below on the next page:

Gannt chart as of 15th September 2021 - Start of project



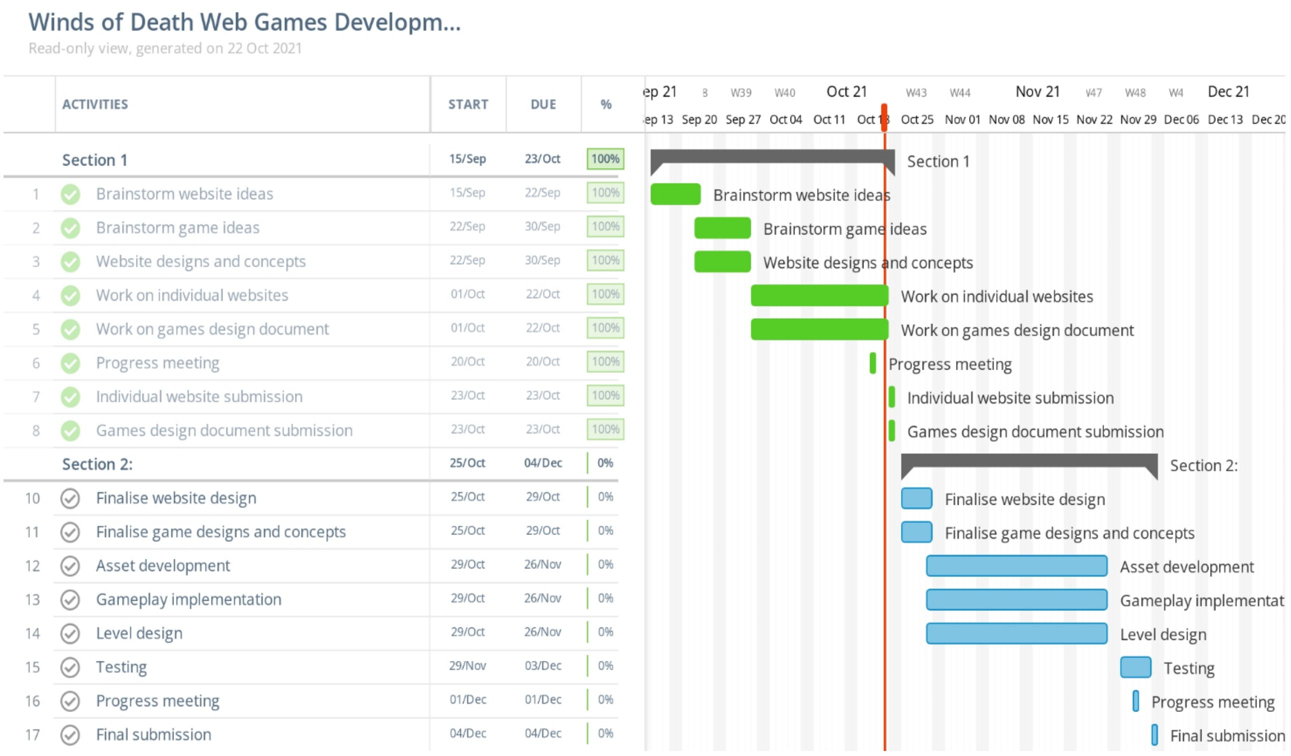
Winds of Death Web Games Developm...

Read-only view, generated on 22 Oct 2021

	ACTIVITIES	START	DUE	%
	Section 1	15/Sep	23/Oct	0%
1	Brainstorm website ideas	15/Sep	22/Sep	0%
2	Brainstorm game ideas	22/Sep	30/Sep	0%
3	Website designs and concepts	22/Sep	30/Sep	0%
4	Work on individual websites	01/Oct	22/Oct	0%
5	Work on games design document	01/Oct	22/Oct	0%
6	Progress meeting	20/Oct	20/Oct	0%
7	Individual website submission	23/Oct	23/Oct	0%
8	Games design document submission	23/Oct	23/Oct	0%
	Section 2:	25/Oct	04/Dec	0%
10	Finalise website design	25/Oct	29/Oct	0%
11	Finalise game designs and concepts	25/Oct	29/Oct	0%
12	Asset development	29/Oct	26/Nov	0%
13	Gameplay implementation	29/Oct	26/Nov	0%
14	Level design	29/Oct	26/Nov	0%
15	Testing	29/Nov	03/Dec	0%
16	Progress meeting	01/Dec	01/Dec	0%
17	Final submission	04/Dec	04/Dec	0%

Sep 21
W39
W40
Oct 21
W43
W44
Nov 21
W47
W48
Dec 21
Sep 13 Sep 20 Sep 27 Oct 04 Oct 11 Oct 18 Oct 25 Nov 01 Nov 08 Nov 15 Nov 22 Nov 29 Dec 06 Dec 13 Dec 20
Section 1
Brainstorm website ideas
Brainstorm game ideas
Website designs and concepts
Work on individual websites
Work on games design document
Progress meeting
Individual website submission
Games design document submission
Section 2:
Finalise website design
Finalise game designs and concepts
Asset development
Gameplay implementat
Level design
Testing
Progress meeting
Final submission

Gannt chart as of 23rd October 2021 - Halfway point of project



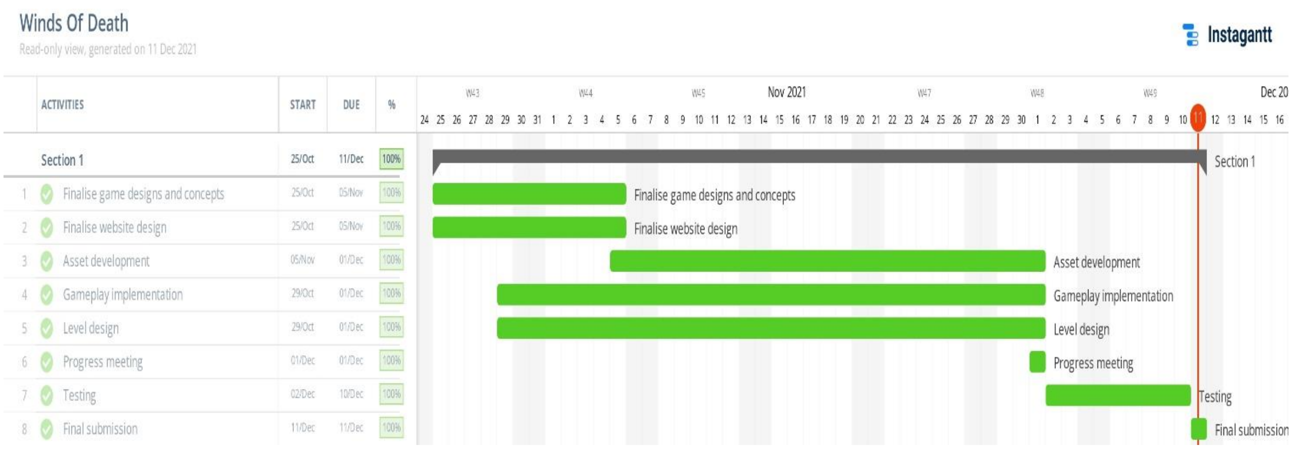
Winds of Death Web Games Developm...

Read-only view, generated on 22 Oct 2021

	ACTIVITIES	START	DUE	%
	Section 1	15/Sep	23/Oct	100%
1	Brainstorm website ideas	15/Sep	22/Sep	100%
2	Brainstorm game ideas	22/Sep	30/Sep	100%
3	Website designs and concepts	22/Sep	30/Sep	100%
4	Work on individual websites	01/Oct	22/Oct	100%
5	Work on games design document	01/Oct	22/Oct	100%
6	Progress meeting	20/Oct	20/Oct	100%
7	Individual website submission	23/Oct	23/Oct	100%
8	Games design document submission	23/Oct	23/Oct	100%
	Section 2:	25/Oct	04/Dec	0%
10	Finalise website design	25/Oct	29/Oct	0%
11	Finalise game designs and concepts	25/Oct	29/Oct	0%
12	Asset development	29/Oct	26/Nov	0%
13	Gameplay implementation	29/Oct	26/Nov	0%
14	Level design	29/Oct	26/Nov	0%
15	Testing	29/Nov	03/Dec	0%
16	Progress meeting	01/Dec	01/Dec	0%
17	Final submission	04/Dec	04/Dec	0%

Sep 21
W39
W40
Oct 21
W43
W44
Nov 21
W47
W48
Dec 21
Sep 13 Sep 20 Sep 27 Oct 04 Oct 11 Oct 18 Oct 25 Nov 01 Nov 08 Nov 15 Nov 22 Nov 29 Dec 06 Dec 13 Dec 20
Section 1
Brainstorm website ideas
Brainstorm game ideas
Website designs and concepts
Work on individual websites
Work on games design document
Progress meeting
Individual website submission
Games design document submission
Section 2:
Finalise website design
Finalise game designs and concepts
Asset development
Gameplay implementat
Level design
Testing
Progress meeting
Final submission

Gannt chart as of 11th December 2021 – Finishing point of project

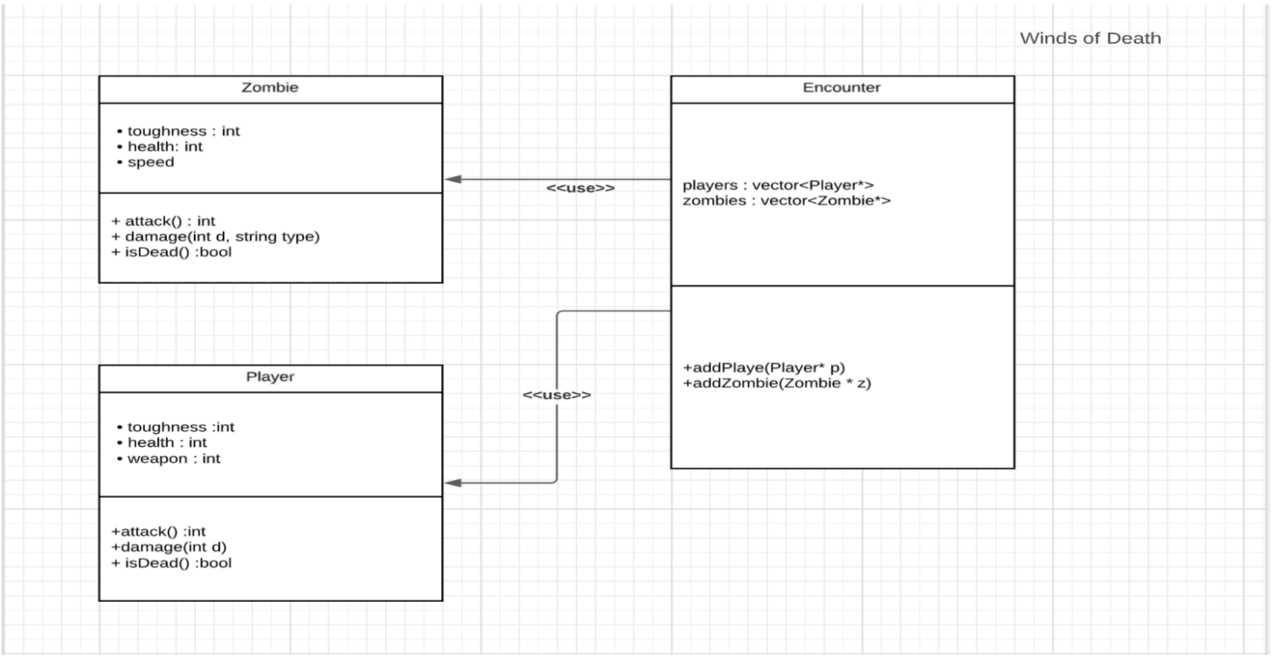

Winds Of Death

Read-only view, generated on 11 Dec 2021

	ACTIVITIES	START	DUE	%
	Section 1	25/Oct	11/Dec	100%
1	Finalise game designs and concepts	25/Oct	05/Nov	100%
2	Finalise website design	25/Oct	05/Nov	100%
3	Asset development	05/Nov	01/Dec	100%
4	Gameplay implementation	29/Oct	01/Dec	100%
5	Level design	29/Oct	01/Dec	100%
6	Progress meeting	01/Dec	01/Dec	100%
7	Testing	02/Dec	10/Dec	100%
8	Final submission	11/Dec	11/Dec	100%

UML Diagram

Below is the UML diagram for our game. It includes the classes for the zombie, player and encounter classes. In addition, it also includes the integers, strings, booleans and variables that we will using in our code.



Roles and responsibilities

Fraser McAulay B00371477

Roles include:

- Level Design
- Environment Art
- Enemy Character Design
- Enemy Character Implementation
- Documentation
- Player Character Implementation
- Player Character Design
- Audio Development
- Concept Art Development
- Story Design
- Character Voices

Mohammed Latif B00333837

Roles include:

- Level Design
- Environment Art
- Enemy Character Design
- Enemy Character Implementation
- Documentation
- Player Character Implementation
- Player Character Design
- Audio Development
- Concept Art Development
- Story Design
- Character Voice

Mihai Maftei B00359190

Roles include:

- Level Design
- Environment Art
- Enemy Character Design
- Enemy Character Implementation
- Documentation
- Player Character Implementation
- Player Character Design
- Audio Development
- Concept Art Development
- Story Design
- Character Voices

Aleksandar Marinov B00351200

Roles include:

- Level Design
- Environment Art
- Enemy Character Design
- Enemy Character Implementation
- Documentation
- Player Character Implementation
- Player Character Design
- Audio Development
- Concept Art Development
- Story Design
- Character Voices

Software Development Lifecycle Methodologies

Throughout the course of this project, we will be adhering to the Agile development methodology. We chose this one as it allows us to rapidly iterate on our design and make changes to the project spec to fit our evolving vision. We will be making use of Scrum, meaning that we will be doing daily iterations on the project, reflecting on those iterations, and then going forward to iterate once more with hurdles in mind. In the same vein, we will hold regular sprint meetings to discuss the state of the project, what we intend to work on next and any difficulties we might encounter in doing so. We considered other development methodologies but settled on agile as it gave us the most freedom. For example, the Waterfall methodology seemed promising, but it requires that each phase be fully complete before we move onto the next with no possibility of going back if we needed to. Agile suited our needs better as Waterfall was not conducive to the fast and responsive style of development we were aiming for.

Schedule

Object 1

Player Characters – Hannibal Lector and Norman Bates

Time Scale:
1 to 3 weeks

• Brainstorming for character ideas, considering character movement system and commence

into implementing the character sprites, mesh and coding.

• Setting up the player character system for moving up, down, left and right. Along the X Axis

and Y Axis

• Finalising character movement system in addition to testing blueprints and code.

Object 2
Player Gun

Time Scale:
1 to 3 weeks

• Brainstorming for character weapons system, considering weapon types and commence into implementing blueprints and coding

• Setting up the weapon system in conjunction with the player movement system

• Finalising the weapon system in addition to testing blueprints and code.

Object 3
Bullet Projectiles

Time Scale:
1 to 3 weeks

• Brainstorming ideas for bullet projectiles, considering various projectiles to apply and commence into implementing blueprints and coding

• Setting up the projectile firing system in conjunction with the player gun and player movement system

• Finalising the projectile firing system in addition to testing blueprints and code.

Object 4
Player Map – City area, Open fields, and Banks of a River

Time Scale:
1 to 3 weeks

• Brain storming ideas for a player map system, considering map variants and commence into implementing a blueprints and coding.

• Setting up the player map in conjunction with the player movement system

• Finalising the player map system, in addition to testing blueprints and code.

Object 5
Player Enemy – Zombies

Time Scale:
1 to 3 weeks

• Brainstorming for character ideas, considering character movement system and commence into implementing the character sprites, mesh, and coding.

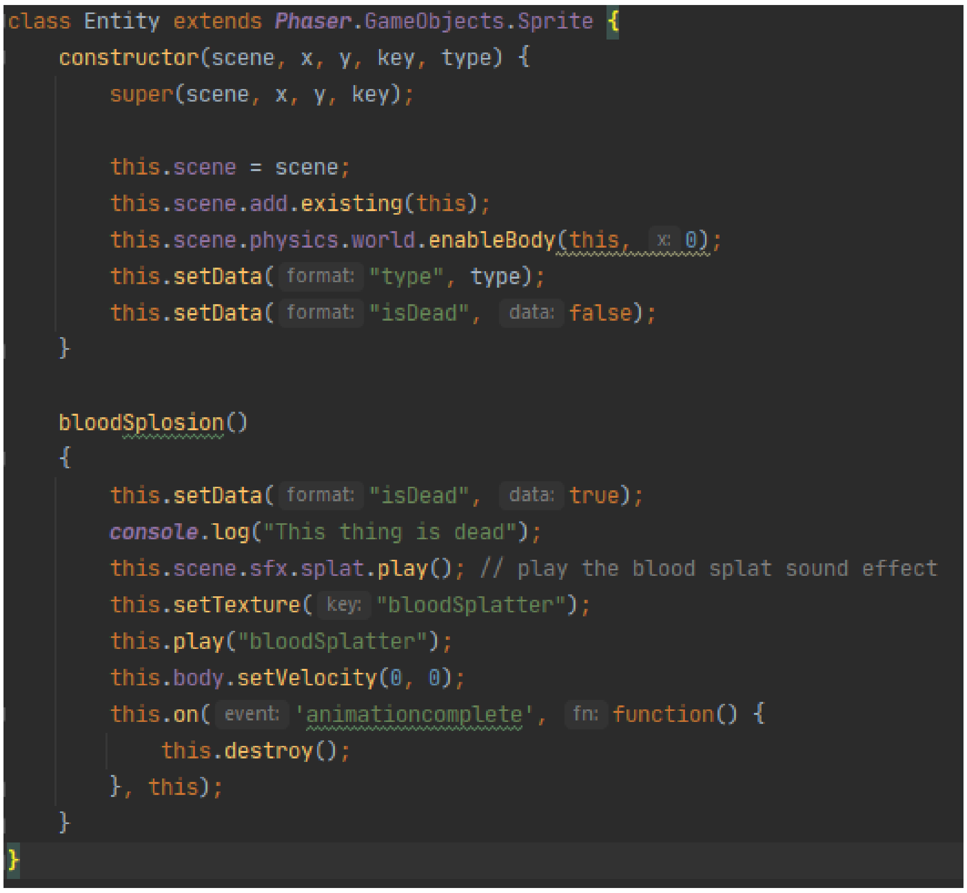
• Setting up the player character system for moving up, down, left and right. Along the X Axis and Y Axis

• Finalising character movement system in addition to testing blueprints and code.

Code examples

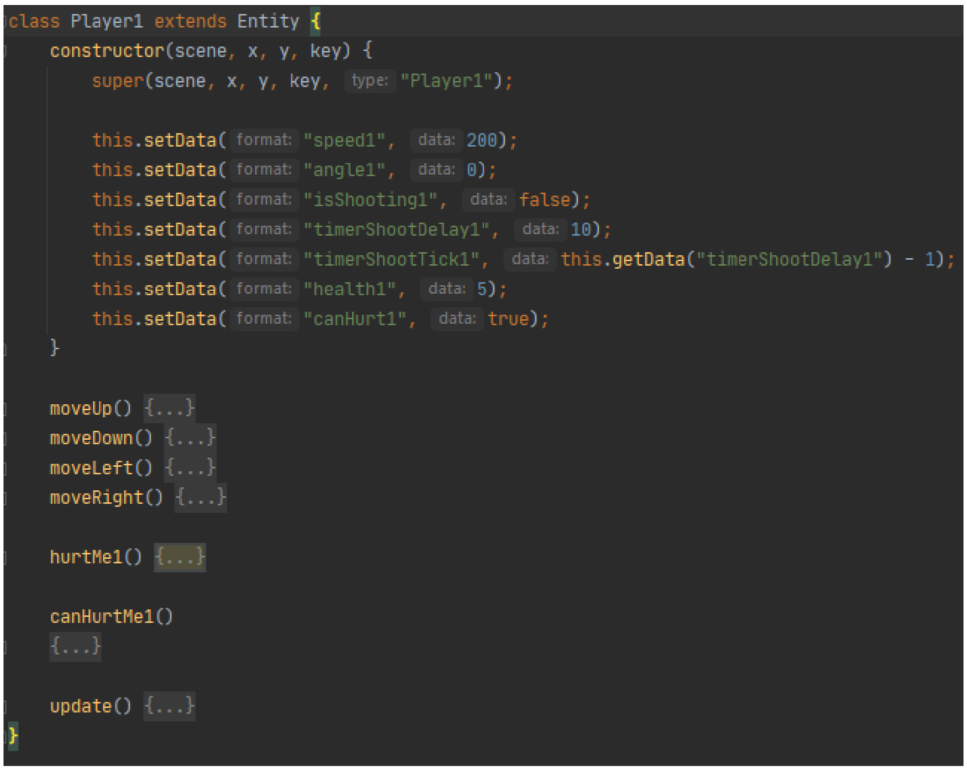
BloodSplosion

The following code below shows how set the entity texture to the blood splatter image, then play the animation. A blood splatter sound effect is played and upon finishing the animation, said entity is deleted from the scene. This code is contained within the definition for an Entity in the game, meaning it can be called on any object.



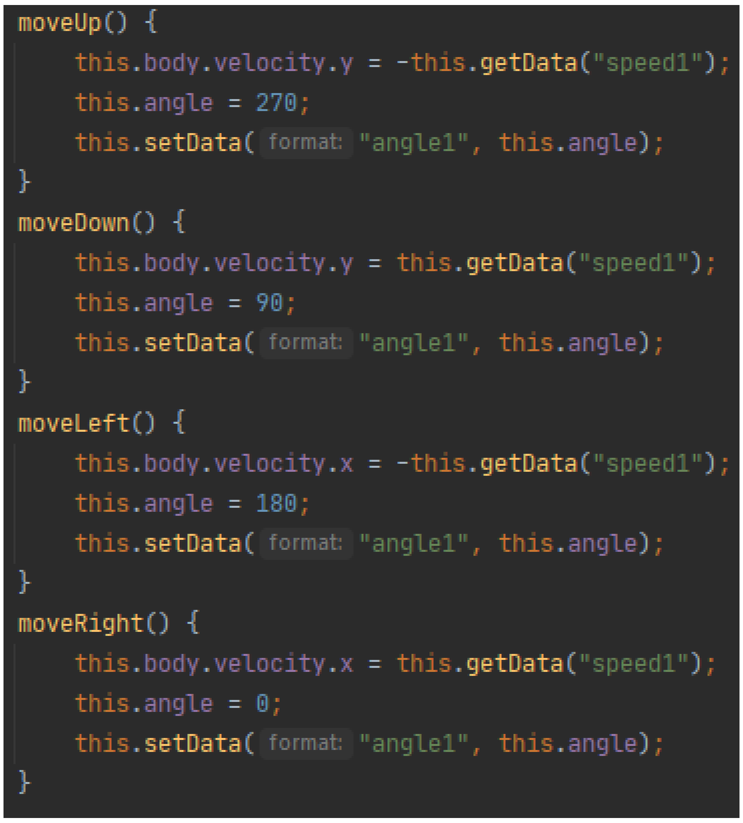
```
class Entity extends Phaser.GameObjects.Sprite {
    constructor(scene, x, y, key, type) {
        super(scene, x, y, key);

        this.scene = scene;
        this.scene.add.existing(this);
        this.scene.physics.world.enableBody(this,  x: 0);
        this.setData( format: "type", type);
        this.setData( format: "isDead",  data: false);
    }

    bloodSplosion()
    {
        this.setData( format: "isDead",  data: true);
        console.log("This thing is dead");
        this.scene.sfx.splat.play(); // play the blood splat sound effect
        this.setTexture( key: "bloodSplatter");
        this.play("bloodSplatter");
        this.body.setVelocity(0, 0);
        this.on( event: 'animationcomplete',  fn: function() {
            this.destroy();
        }, this);
    }
}
```

Player



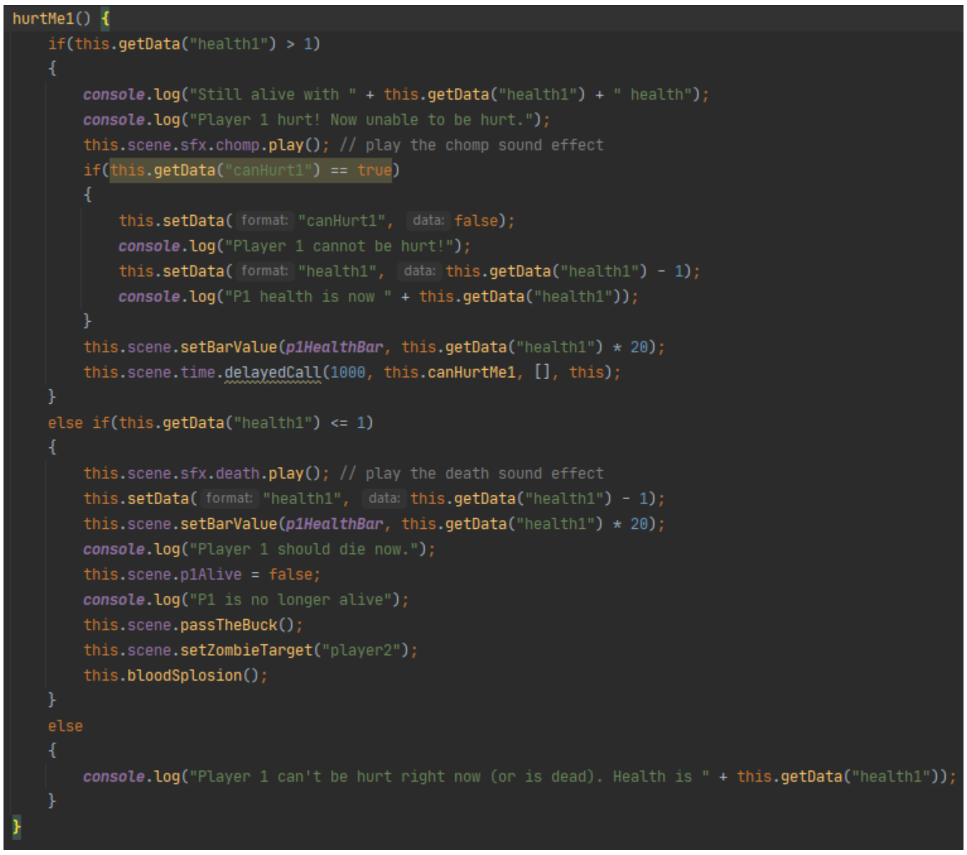
```
class Player1 extends Entity {
    constructor(scene, x, y, key) {
        super(scene, x, y, key, type: "Player1");

        this.setData( format: "speed1",  data: 200);
        this.setData( format: "angle1",  data: 0);
        this.setData( format: "isShooting1",  data: false);
        this.setData( format: "timerShootDelay1",  data: 10);
        this.setData( format: "timerShootTick1",  data: this.getData("timerShootDelay1") - 1);
        this.setData( format: "health1",  data: 5);
        this.setData( format: "canHurt1",  data: true);
    }

    moveUp() {...}
    moveDown() {...}
    moveLeft() {...}
    moveRight() {...}

    hurtMe1() {...}

    canHurtMe1()
    {...}

    update() {...}
}
```

The following code shows our player controls and how the play sprite moves up, down, left and right throughout the X axis and Y axis. There is also code that handles the player getting hurt and eventually dying, with player 1 having special code to pass camera control over to player 2 on death.



```
moveUp() {
    this.body.velocity.y = -this.getData("speed1");
    this.angle = 270;
    this.setData( format: "angle1", this.angle);
}
moveDown() {
    this.body.velocity.y = this.getData("speed1");
    this.angle = 90;
    this.setData( format: "angle1", this.angle);
}
moveLeft() {
    this.body.velocity.x = -this.getData("speed1");
    this.angle = 180;
    this.setData( format: "angle1", this.angle);
}
moveRight() {
    this.body.velocity.x = this.getData("speed1");
    this.angle = 0;
    this.setData( format: "angle1", this.angle);
}
```

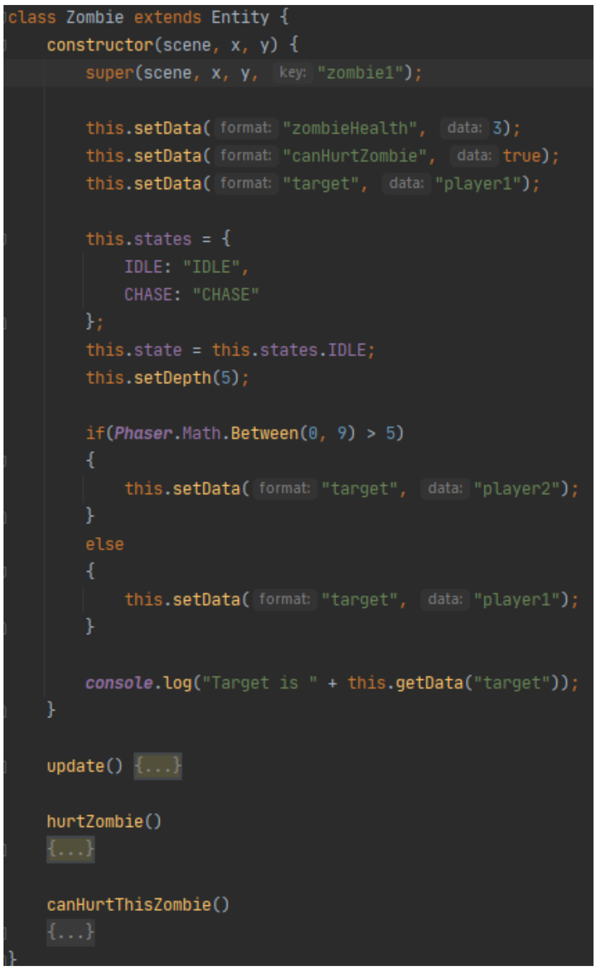
The following code shows how player health is handled when coming into contact with an enemy. Upon contact, the player loses 1 point of health which is promptly reflected in the health bar in the current scene. At the same time, the “cantHurt” boolean is set to **false** with a delayed call being set to turn it **true** after 1 second. Should the player have 1 health point left when hit, the game will play a sound effect to signify their death, tell the current scene that player 1 is dead, set every zombie to target the other player and finally call the *bloodSplosion* function which is built into every entity in the *entities.js* file. Since the first player has control of the camera, a function (passTheBuck) is called within the given scene to make the camera start following player 2 so long as they are still alive.



```
hurtMe1() {
    if(this.getData("health1") > 1)
    {
        console.log("Still alive with " + this.getData("health1") + " health");
        console.log("Player 1 hurt! Now unable to be hurt.");
        this.scene.sfx.chomp.play(); // play the chomp sound effect
        if(this.getData("canHurt1") == true)
        {
            this.setData( format: "canHurt1",  data: false);
            console.log("Player 1 cannot be hurt!");
            this.setData( format: "health1",  data: this.getData("health1") - 1);
            console.log("P1 health is now " + this.getData("health1"));
        }
        this.scene.setBarValue(p1HealthBar, this.getData("health1") * 20);
        this.scene.time.delayedCall(1000, this.canHurtMe1, [], this);
    }
    else if(this.getData("health1") <= 1)
    {
        this.scene.sfx.death.play(); // play the death sound effect
        this.setData( format: "health1",  data: this.getData("health1") - 1);
        this.scene.setBarValue(p1HealthBar, this.getData("health1") * 20);
        console.log("Player 1 should die now.");
        this.scene.p1Alive = false;
        console.log("P1 is no longer alive");
        this.scene.passTheBuck();
        this.scene.setZombieTarget("player2");
        this.bloodSplosion();
    }
    else
    {
        console.log("Player 1 can't be hurt right now (or is dead). Health is " + this.getData("health1"));
    }
}
```

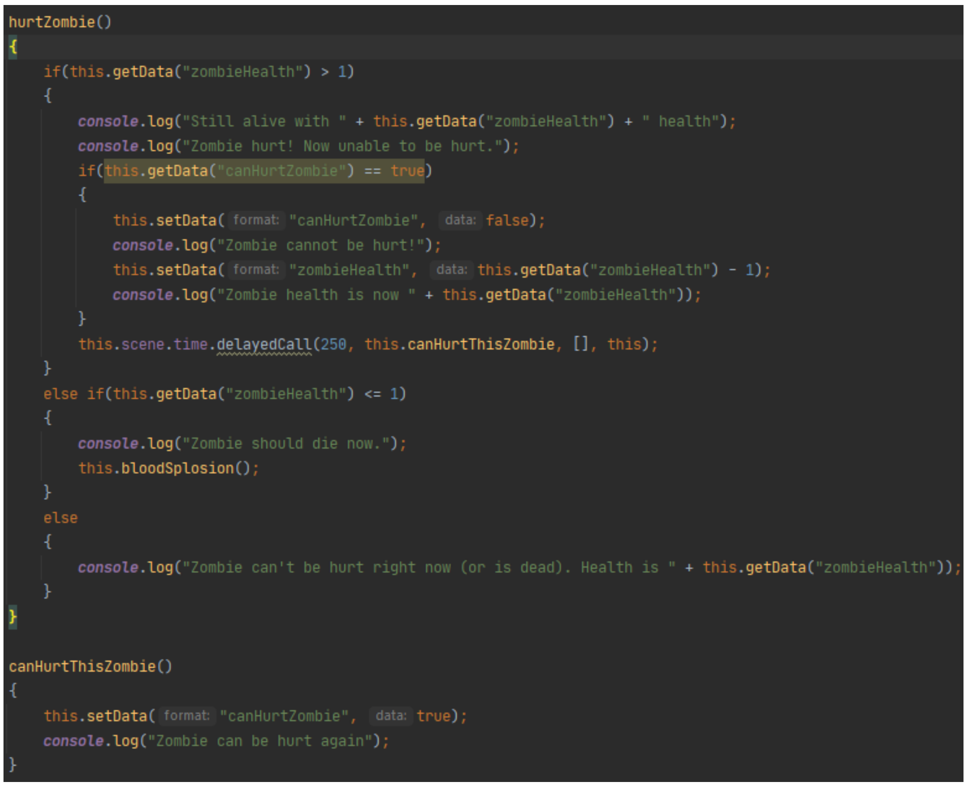
Zombie

The following code below shows our Zombie class. This class loads our zombie sprites and code for their movements. It starts by setting data for health, whether it can be hurt and a default target value of “player1” which is then randomised to choose either the first or second player as its target. Next the two states are defined: Idle and Chase. When the target player reaches a certain threshold, the zombie will start facing them and moving closer to deal damage.



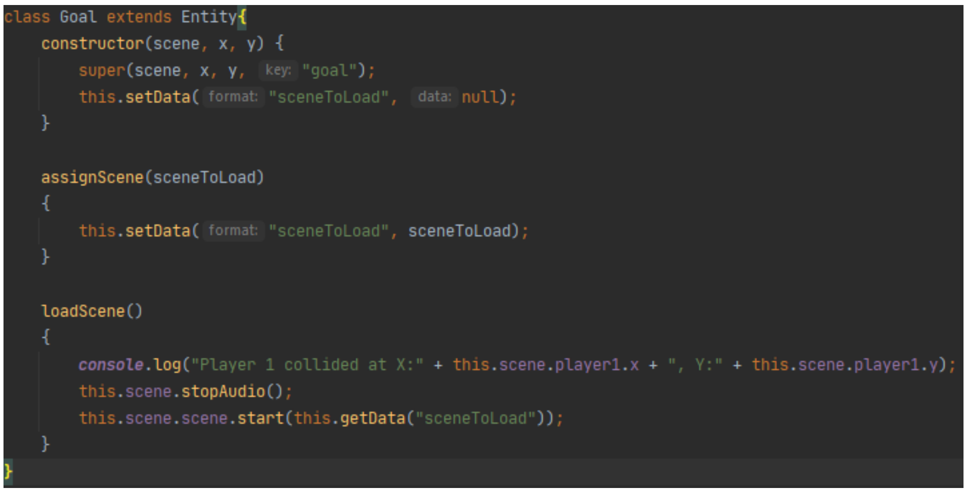
```
class Zombie extends Entity {
    constructor(scene, x, y) {
        super(scene, x, y,  key: "zombie1");

        this.setData( format: "zombieHealth",  data: 3);
        this.setData( format: "canHurtZombie",  data: true);
        this.setData( format: "target",  data: "player1");

        this.states = {
            IDLE: "IDLE",
            CHASE: "CHASE"
        };
        this.state = this.states.IDLE;
        this.setDepth(5);

        if(Phaser.Math.Between(0, 9) > 5)
        {
            this.setData( format: "target",  data: "player2");
        }
        else
        {
            this.setData( format: "target",  data: "player1");
        }

        console.log("Target is " + this.getData("target"));
    }

    update() {...}

    hurtZombie()
    {...}

    canHurtThisZombie()
    {...}
}
```

The code that handles zombies taking damage and dying is essentially the same as that of the code for player 1 and 2, albeit with a shorter delay on invincibility.



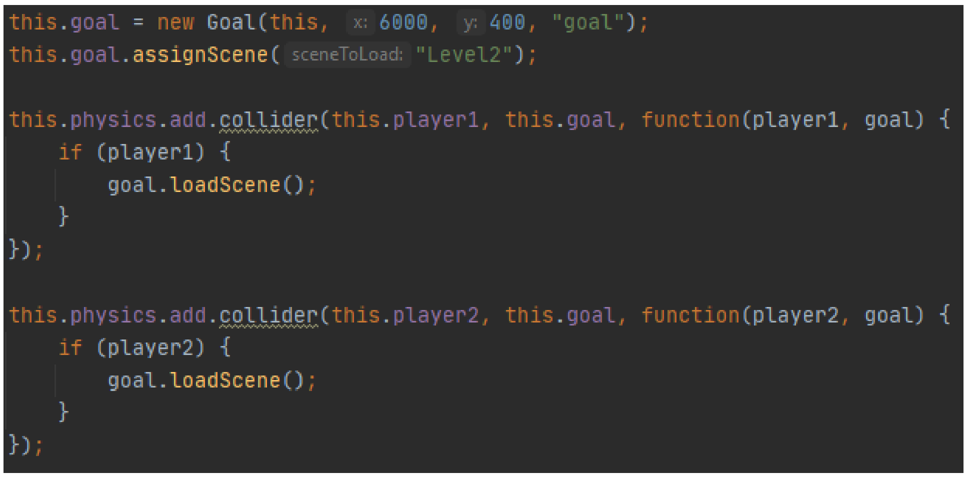
```
hurtZombie()
{
    if(this.getData("zombieHealth") > 1)
    {
        console.log("Still alive with " + this.getData("zombieHealth") + " health");
        console.log("Zombie hurt! Now unable to be hurt.");
        if(this.getData("canHurtZombie") == true)
        {
            this.setData( format: "canHurtZombie",  data: false);
            console.log("Zombie cannot be hurt!");
            this.setData( format: "zombieHealth",  data: this.getData("zombieHealth") - 1);
            console.log("Zombie health is now " + this.getData("zombieHealth"));
        }
        this.scene.time.delayedCall(250, this.canHurtThisZombie, [], this);
    }
    else if(this.getData("zombieHealth") <= 1)
    {
        console.log("Zombie should die now.");
        this.bloodSplosion();
    }
    else
    {
        console.log("Zombie can't be hurt right now (or is dead). Health is " + this.getData("zombieHealth"));
    }
}

canHurtThisZombie()
{
    this.setData( format: "canHurtZombie",  data: true);
    console.log("Zombie can be hurt again");
}
```

Goal



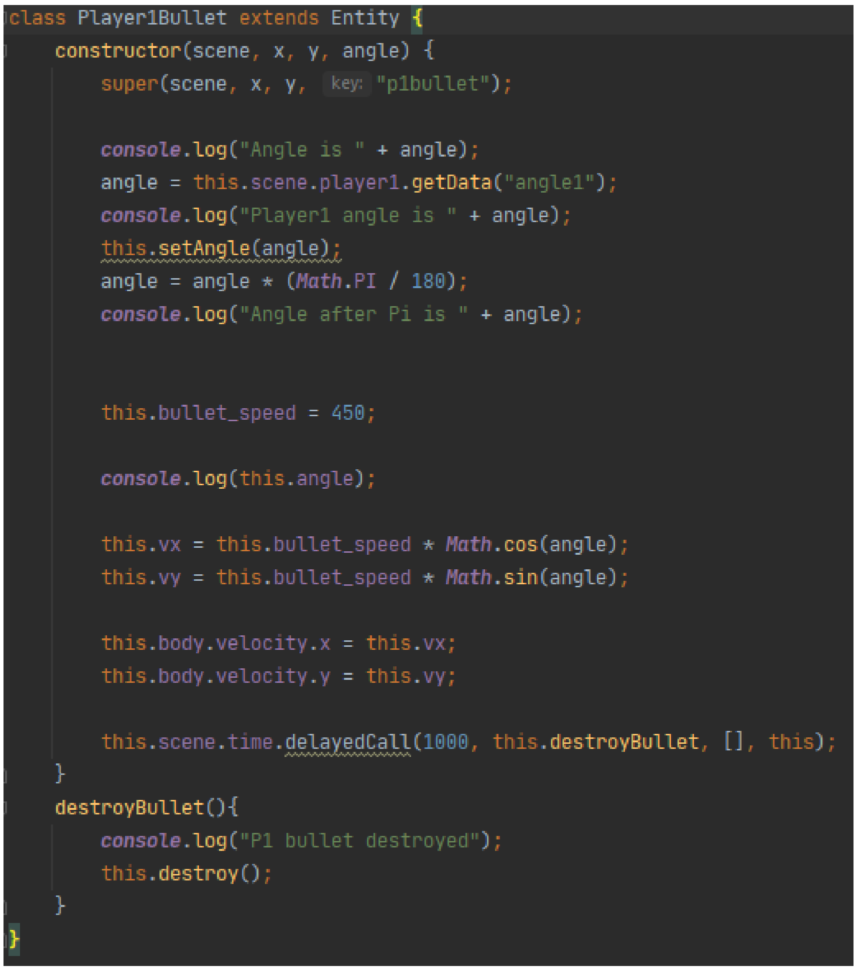
```
class Goal extends Entity{
    constructor(scene, x, y) {
        super(scene, x, y, key: "goal");
        this.setData( format: "sceneToLoad",  data: null);
    }

    assignScene(sceneToLoad)
    {
        this.setData( format: "sceneToLoad", sceneToLoad);
    }

    loadScene()
    {
        console.log("Player 1 collided at X:" + this.scene.player1.x + ", Y:" + this.scene.player1.y);
        this.scene.stopAudio();
        this.scene.scene.start(this.getData("sceneToLoad"));
    }
}
```

To get from one level to the next, goal objects were written to handle level loading. They are created with a “sceneToLoad” value which is set upon creation within any particular scene. When either player comes into contact, all audio in the game is stopped and the next scene is loaded using the assigned value.



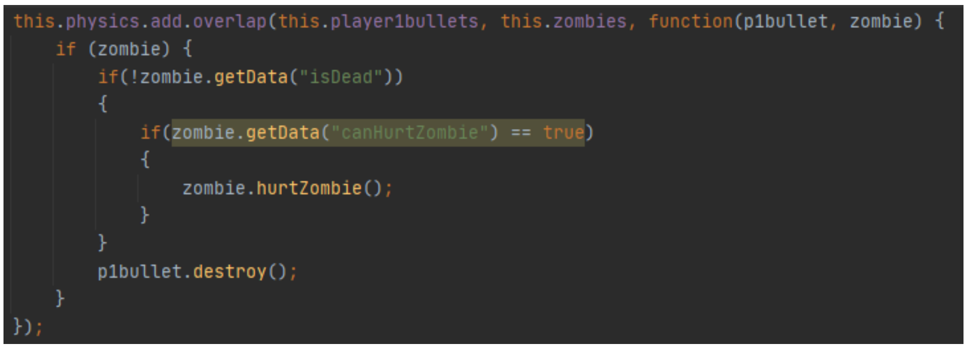
```
this.goal = new Goal(this,  x: 6000,  y: 400, "goal");
this.goal.assignScene( sceneToLoad: "Level2");

this.physics.add.collider(this.player1, this.goal, function(player1, goal) {
    if (player1) {
        goal.loadScene();
    }
});

this.physics.add.collider(this.player2, this.goal, function(player2, goal) {
    if (player2) {
        goal.loadScene();
    }
});
```

Bullets

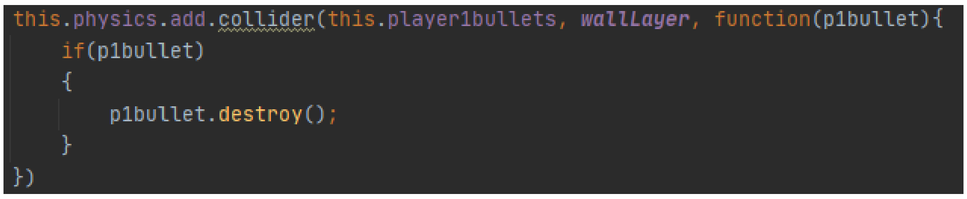


```
class Player1Bullet extends Entity {
    constructor(scene, x, y, angle) {
        super(scene, x, y,  key: "p1bullet");

        console.log("Angle is " + angle);
        angle = this.scene.player1.getData("angle1");
        console.log("Player1 angle is " + angle);
        this.setAngle(angle);
        angle = angle * (Math.PI / 180);
        console.log("Angle after Pi is " + angle);


        this.bullet_speed = 450;

        console.log(this.angle);

        this.vx = this.bullet_speed * Math.cos(angle);
        this.vy = this.bullet_speed * Math.sin(angle);

        this.body.velocity.x = this.vx;
        this.body.velocity.y = this.vy;

        this.scene.time.delayedCall(1000, this.destroyBullet, [], this);
    }
    destroyBullet(){
        console.log("P1 bullet destroyed");
        this.destroy();
    }
}
```

When player bullets are created, they get the data for the player's current angle then use it to determine which way they should be facing along with the direction they should travel. They also start a delayed call to destroy themselves after 1 second. In each level, collisions are set up with the zombies so that the bullets call their hurt function upon contact and then destroy. They are also set up to destroy upon touching any of the walls in the level.



```
this.physics.add.overlap(this.player1bullets, this.zombies, function(p1bullet, zombie) {
    if (zombie) {
        if(!zombie.getData("isDead"))
        {
            if(zombie.getData("canHurtZombie") == true)
            {
                zombie.hurtZombie();
            }
        }
        p1bullet.destroy();
    }
});
```



```
this.physics.add.collider(this.player1bullets, wallLayer, function(p1bullet){
    if(p1bullet)
    {
        p1bullet.destroy();
    }
})
```

GitHub Link

We will be utilising Microsoft OneDrive to manage the team's version control and code sharing. The Microsoft OneDrive page is linked here: https://bit.ly/3EKy8bp.

Version 1.0

In the future versions of the game, we would like to add the following:

- Attacking abilities such as increasing and upgrading the damage that the players weapons cause and increasing or decreasing the firing rate of their weapons. In addition to implementing a weapon pickup system.
- Defensive abilities such as upgrading the resistance of player shields and implementing a health pickup system.
- Pickups that restore player health upon contact.
- Future additions may include the addition of more game levels. There would also be a choice in the options scene for setting the level of difficulty for the game.

On reflection, we are proud of how the game turned out as it was a special challenge to develop it. The above features were planned but cut when we realised that we wouldn't have enough time to add and properly test them.

References

Agile Game Development – A Quick Overview - Marionette Studio (2016)

https://marionettestudio.com/agile-game-development-quick-overview/

Game-Scrum: An Approach to Agile Game Development - Andre Godoy, Ellen F. Barbosa (2010)

http://sbgames.org/papers/sbgames10/computing/short/Computing_short19.pdf

Java Singleton Design Pattern Best Practices with Examples (2013)

https://www.journaldev.com/1377/java-singleton-design-pattern-best-practices-examples

Top 4 software development methodologies – Synopsys (2017)

https://www.synopsys.com/blogs/software-security/top-4-software-development-methodologies/