Technical Report detailing the design process used to develop the 2D top-down shooter video game *BlockShot*

Abstract

This document is a technical report that outlines the iterative design process that was incorporated for the planning, development and testing of the game *BlockShot.*

BlockShot is a two-dimensional, top-down shooter in which the player moves along a circular path around a central gun with the primary objective of destroying the gun before it reduces the players health total to zero.

Introduction

The creation of *BlockShot* was limited by some primary requirements and constraints. The primary requirements entailed the inclusion of:

- a central gun around which the player moves along a circular path, as well as
- a combat system that allows the player to fire projectiles at the central gun and vice versa.

The primary constraint was time, as *BlockShot* had to be developed over the course of only a few weeks and other exams had to be written during those weeks.

Section 1 of this report explores the planning and development of *BlockShot*, Section 2 looks at how the game was tested and the feedback that was received and Section 3 is the conclusion.

Section 1: Planning and Development

The first step in the development process was to identify the core aspects of the game and plan exactly how these aspects would be coded and implemented into the game.

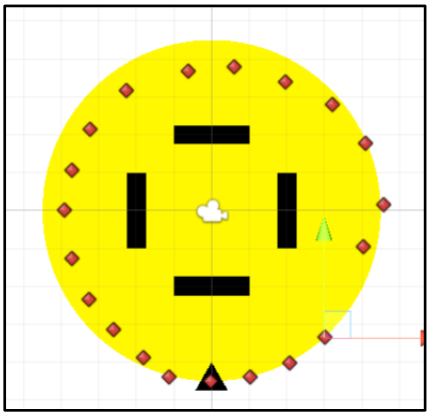

Figure 1

The first major aspect was movement. It was specified that the player would have to move in a circle around the central gun, this movement was different from most other games as it was not linear, so the usual principles of translation along a single axis could not be incorporated. The solution to this problem was the use of waypoints [1], which allowed for the specification of a path along which the player would move. To ensure the waypoints were placed in a perfect circle, a circle sprite was placed in the scene and then traced out as can be seen in Figure 1.

The code shown in Figure 2 was used to move the player to the next waypoint, once the last waypoint was reached, it moved back to the first one, and hence a circular movement path was created.

```
private void Move1()
{
    if (waypointIndex <= waypoints.Length -1)
    {   transform.position = Vector2.MoveTowards(transform.position,
            waypoints[waypointIndex].transform.position,
            moveSpeed*2*Time.deltaTime
            );

        if (transform.position == waypoints[waypointIndex].transform.position)
        { waypointIndex += 1; }
        if (waypointIndex==27)
        {waypointIndex = 0;}}}
```

Figure 2

```
void faceMouse()
{ Vector3 mousePosition = Input.mousePosition;
    mousePosition = Camera.main.ScreenToWorldPoint(mousePosition);

    Vector2 direction = new Vector2(
        mousePosition.x - transform.position.x,
        mousePosition.y - transform.position.y
        );
    transform.up = direction;

    if (timeBtwShots<=0 ) {

        if (Input.GetKeyDown(KeyCode.Space))
        {Instantiate(projectile, shotPoint.position, transform.rotation);
            timeBtwShots = startTimeBtwShots;}
    }else
    {timeBtwShots -= Time.deltaTime;}}
```

Figure 3

The next major aspect of the game that had to be developed was the combat system. The primary objective of the combat system included a way for the player to aim and shoot projectiles. The first part of the code seen in Figure 3 is used rotate the player game object to face the mouse's position onscreen, allowing them to aim. The second part of the code is what enables them to fire projectiles at a predetermined rate.

The other aspect the combat system had to include, was a way for the gun in the center to fire projectiles at the player. The code that allowed the gun to follow the players position and fire projectiles, was similar to the code used to rotate the player game object towards the mouse's position on screen and fire projectiles . One problem that became evident, was the player's position at the time the gun fired a projectile was not the same as when the projectile eventually reached the player's movement path. The solution to this was, to incorporate code that made the gun's projectiles similar to homing missiles, in the sense that they would follow the player until they were destroyed or collided with the player. The code in Figure 4 is what allowed these projectiles to follow the player's dynamic position and act like homing missiles. Collision detection for both the player's and the gun's projectiles were done using Rays and RayCasting[2].

```
void Update()
{transform.position = Vector2.MoveTowards(transform.position, obj.transform.position, speed*Time.deltaTime);
    if (transform.position== obj.transform.position)
    {obj.GetComponent<FaceMouse>().TakeDamage(damage);
        Destroy(gameObject);}}}
```

Figure 4

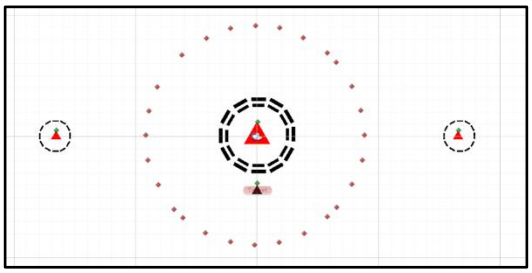

Figure 5

Once the combat and movement systems were developed, several levels could be developed for the game by altering the number of enemy guns in a scene, the layout of these guns, their rate of fire as well as the number of barrier layers around them. Figure 5 depicts a level of the final game in which three guns have been used, two with higher fire rates and one central gun with a low fire rate but a projectile that does a lot of damage.

The finishing touches for the game included the creation of a home menu, a pause menu and level select, all of which were done using the Unity Engine UI editor. Minimal code was necessary as the SetActive() method [3] was used to switch between the menus.

Section 2: Play Testing and Feedback

The play testing of the game was an iterative, multi-level process. Initially play testing was done extensively by the developer, with minor bugs being fixed and game balancing being done along the way. Next the game was play tested by numerous people with varying skill and experience with video games. The general feedback was that the projectiles that were shot by the guns moved too quickly to dodge or destroy and that the combat system had to be enhanced as just pressing the Space bar to fire got repetitive and boring quickly. Fixing the first problem was easy and required adjusting the size and speed at which the projectiles travelled. The second problem was a bit more challenging to resolve. After experimenting with various ideas, the best solution was to incorporate a “Power Up” that would charge as you played and then could be activated by the player, allowing them to rapidly fire projectiles for a short duration of time by holding in the “X” key.

Section 3: Conclusion

Comparing the requirements that were set out in the Introduction and the final iteration of the game, it is evident that all the primary requirements were met and that *BlockShot* is a challenging, yet enjoyable game. Some future improvement that could be made is the inclusion of different:

- weapons that the player could equip with different fire rates and damage statistics
- types of enemy guns that could appear in the levels.

These new weapons and enemies would add a new element to the combat system and allow innovative combat challenges that the player would have to strategically overcome.

With the inclusion of these future improvements, *BlockShot* would come to life as a fully-fledged top-down shooter game.

References

[1] - Handverger, M. (2019). *Building a Waypoint Pathing System In Unity | Tricky Fast Studios*. [online] Trickyfast.com. Available at: http://www.trickyfast.com/2017/09/21/building-a-waypoint-pathing-system-in-unity/ [Accessed 6 Jun. 2019].

[2] - Technologies, U. (2019). *Unity - Scripting API: Physics.Raycast*. [online] Docs.unity3d.com. Available at: https://docs.unity3d.com/ScriptReference/Physics.Raycast.html [Accessed 7 Jun. 2019].

[3] - Technologies, U. (2019). *Unity - Scripting API: GameObject.SetActive*. [online] Docs.unity3d.com. Available at: https://docs.unity3d.com/ScriptReference/GameObject.SetActive.html [Accessed 7 Jun. 2019].