Game Design Document

Firas El-Jerdy

Notre Dame University

May 18, 2018

Game Design Document

Note: I decided to change up the way I write my documents, that is because I felt that I wasn't completely letting you know what I had done in my projects.

Introduction:

I have created an endless runner, side scrolling 2D game in Unity. A character runs on random platforms thrown at him, with the incentive of picking up power ups and coins. The setting of the game is set in snowy surroundings, with the clear indication of the snow covered platforms, hail storm, snowmen, and the dark weather.

Pooling system:

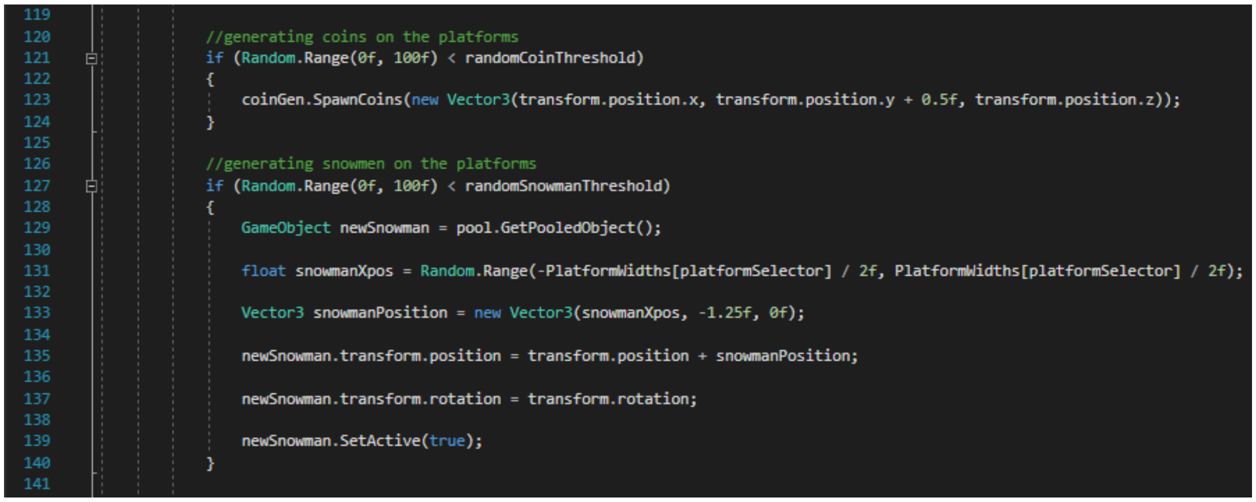
I have created a pooling class that can take in any asset directly. First a gameObject is created, object that want to be pooled is stored inside of it, the empty gameObject is then given the pool manager class. Then specific classes are made to shape the scenes behaviour.

Take for example, the coins. The coins have a class specifically to manage how they should be placed, three coins are always placed on a platform with specific width between them and a fixed height. Then, the class platform manager is responsible for creating randomness. In the case of the coins, there is a ½ chance of any given platform passing on the screen to get 3 coins.

The next example are the actual platforms themselves. 4 Types of platforms are prefabs and pooled in the system. 2 points are placed, one on the right side of the camera; that is responsible for reusing the platforms, and one on the left back side with the intent for reclaiming platforms. So, as the camera moves the points will move alongside them, and objects are put back in and pulled out of the system (That system is used for every other object not just platforms). Platforms are given a random height between them (another point moves with the camera to determine the maximum height a platform can attain). And a random width determined by a minimum width and maximum width variables supplied by me.

Next, are the Powerups which are pooled and fed to the pooling system. Their behaviour is also determined by the class called platform generator, which restricts their behaviour to spawning between 2 platforms, with 0.15 chance of appearing at any given platform.

Lastly, the snowman (obstacle). One type of obstacle appears with a chance of 0.35. The obstacle is placed randomly over the width of the platform.

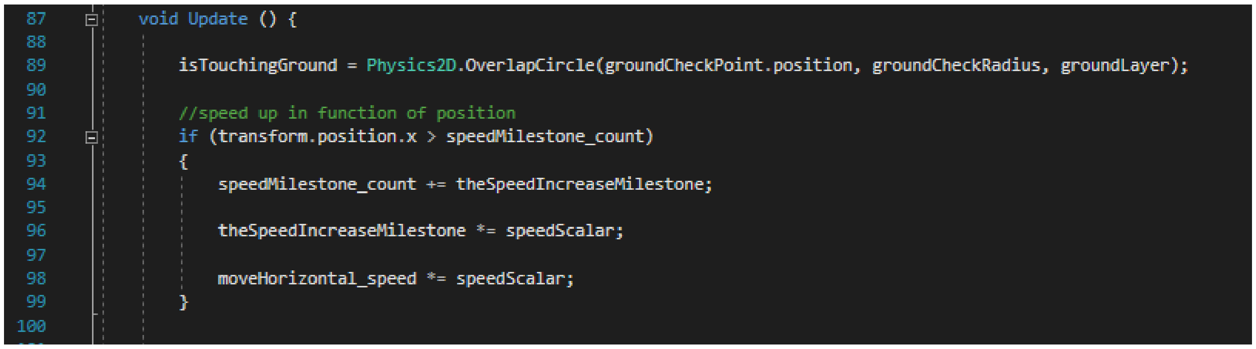


```csharp

//generating coins on the platforms
if (Random.Range(0f, 100f) < randomCoinThreshold)
{
    coinGen.SpawnCoins(new Vector3(transform.position.x, transform.position.y + 0.5f, transform.position.z));
}

//generating snowmen on the platforms
if (Random.Range(0f, 100f) < randomSnowmanThreshold)
{
    GameObject newSnowman = pool.GetPooledObject();

    float snowmanXpos = Random.Range(-PlatformWidths[platformSelector] / 2f, PlatformWidths[platformSelector] / 2f);

    Vector3 snowmanPosition = new Vector3(snowmanXpos, -1.25f, 0f);

    newSnowman.transform.position = transform.position + snowmanPosition;

    newSnowman.transform.rotation = transform.rotation;

    newSnowman.SetActive(true);
}

```

Above is an example of the random functions.

Mechanics:

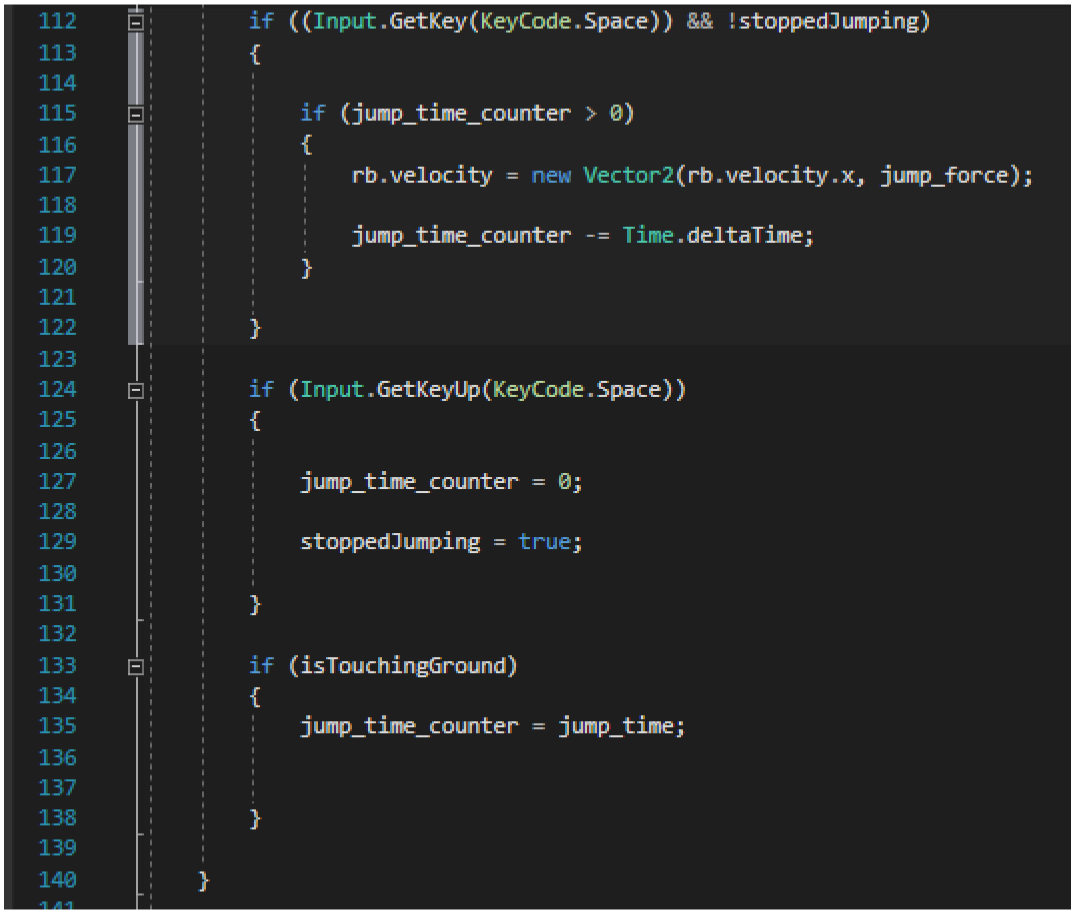
The player runs on the x-axis indefinitely at a fixed movement speed. The more the player crosses distance, the higher the speeds. That is defined by a velocity at first, and a point placed every specific unit of measurement in the world, also known as a milestone. Each time the character gets through a milestone, speed vector is multiplied by a scalar of 1.025. Allowing the character to speed up in function of distance.



```
void Update () {

    isTouchingGround = Physics2D.OverlapCircle(groundCheckPoint.position, groundCheckRadius, groundLayer);

    //speed up in function of position
    if (transform.position.x > speedMilestone_count)
    {
        speedMilestone_count += theSpeedIncreaseMilestone;

        theSpeedIncreaseMilestone *= speedScalar;

        moveHorizontal_speed *= speedScalar;
    }

```

Allowing the game to become harder, to a certain speed.

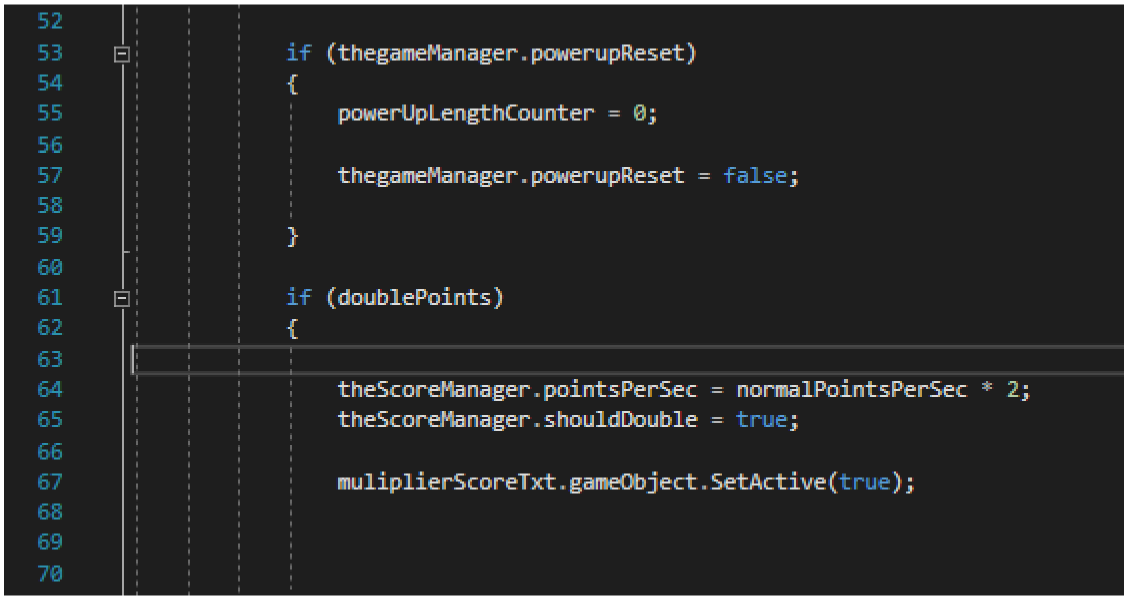
To further add, the character is capable of jumping. The intensity of the jump is specified by the actual mass of the character, the gravity scale, and the upward velocity given to the character. The character jumps with the SPACE bar. The player can specify the intensity of the jump by how much the SPACE bar is pressed. The longer the press the more angular and upward momentum is exerted on the character.



```
        if ((Input.GetKey(KeyCode.Space)) && !stoppedJumping)
        {

            if (jump_time_counter > 0)
            {
                rb.velocity = new Vector2(rb.velocity.x, jump_force);

                jump_time_counter -= Time.deltaTime;
            }

        }

        if (Input.GetKeyUp(KeyCode.Space))
        {

            jump_time_counter = 0;

            stoppedJumping = true;

        }

        if (isTouchingGround)
        {
            jump_time_counter = jump_time;


        }

    }
```

Game over is prompted when and if the character collides with an element tagged with "RestartBox". A quad with no mesh rendered is placed under the character and moves in accordance with the camera controller on the x-axis. Another element is the obstacle, given a UI graphic. Once the character hits any of them, a menu is prompted signalling the loss and giving the player a choice between retrying and quitting to the main menu. Retrying or restarting the game, resets the player to the original starting position, alongside all the points, score and behaviours.

There exists 2 possible powerups, they are generated randomly between platforms with either 2 fucntions: Double score, Safe mode. Double score increases the score intake by 2 for a time of 10 seconds, and safe mode removes all nearby obstacles for a period of 10 seconds. They are picked up by the player and used instantly.



```

                if (thegameManager.powerupReset)
                {
                    powerUpLengthCounter = 0;

                    thegameManager.powerupReset = false;

                }

                if (doublePoints)
                {

                    theScoreManager.pointsPerSec = normalPointsPerSec * 2;
                    theScoreManager.shouldDouble = true;

                    muliplierScoreTxt.gameObject.SetActive(true);



```

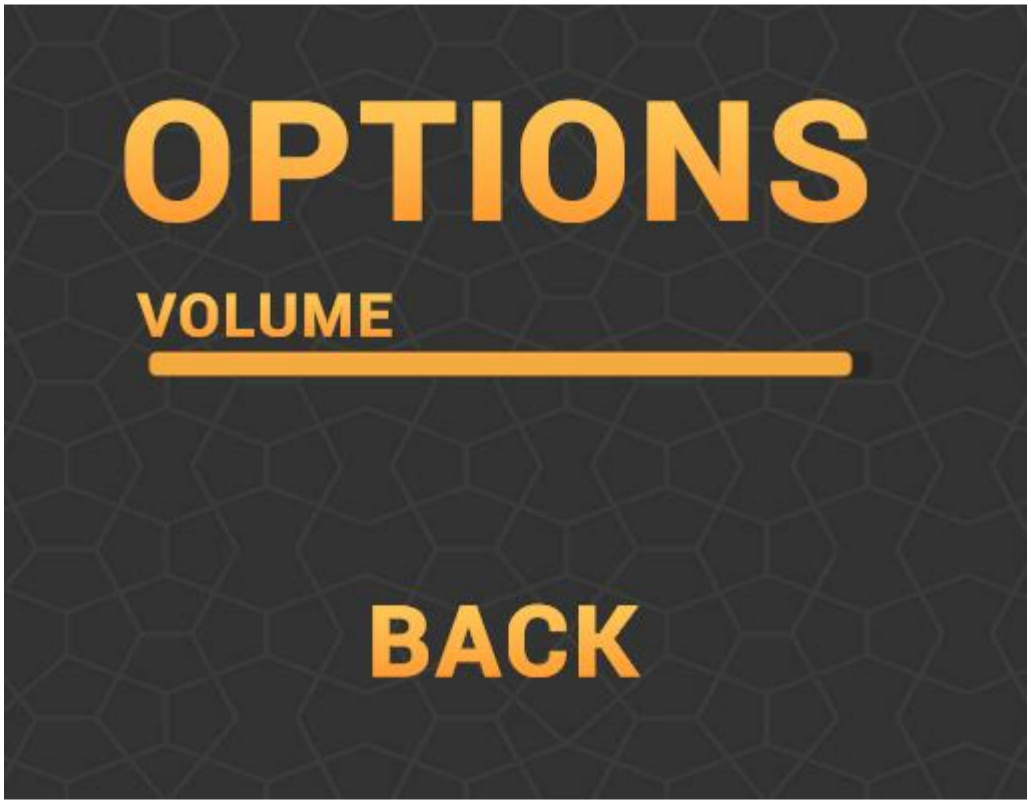
The player is then prompted on screen of the active powerup.

Scoring system:

The score is correlated to time played, and coins. The more the player survives and coins picked up the higher the score. Score is given at fixed rate (5 points per frame, that around 1/fps * 5). A coin gives 100 points. On the top left of the screen there is the current scored displayed, top right is the all-time high score achieved on the system.

Navigation:

At first, when the game is launched. A menu is presented, letting the player either playing the game, entering the settings, and or quitting the game.



The main mixer volume can be increased and decreased from the setting menu.

The pause menu is initiated by the ESC button on the keyboard, opening a window to go back to menu or restart the current session. Time and velocity are frozen in the meantime.

Assets:

Powerups: PowerUp Particles by MHLab.

Platforms: 2D Platformer Winter by Losev Valera

Character: 2D archers sprite by HONETI.

Particle system: Hail Particles Pack by Luke Peek.

Coins: Animated 2D coins by Allastar.

Music: Main music: Happy Upbeat Background Music from MorningLightMusic

Main menu music: GAME MENU found on Soundimage.org/fantasywonder/

Thank you!