2/5/2018

Game Specification Document

Prototype name: Mining Life

Mojtaba Hafezi
ID: N0771021

Contents

Bibliography 1
Brief description: 2
Gameplay description: 2
Screen Images: 3
Implementation: 5

Figure 1 Main menu 3
Figure 2 Cave instance 3
Figure 3 Instance from far away – currently only 4 tiles are in use 4
Figure 4 End of the instance - when player gets close then new terrain gets generated 4
Figure 5 Player mining downwards 4
Figure 6 Initialise method in Board Manager 5
Figure 7 Player character with 3 colliders 6

Bibliography

Jackson, S. (2014) *'Mastering Unity 2D Game',* Birmingham: Packt Publishing Ltd.

Pereira, V. (2014) *'Learning Unity 2D Game Development by Example',* Birmingham: Packt Publishing Ltd.

Watkins, R. (2016) *'Procedural Content Generation for Unity',* Birmingham: Packt Publishing Ltd..

Brief description:

Mining Life takes you deep under the surface to mine for valuable resources and makes you look for hidden gems and riches. The game includes a fast-paced gameplay with exciting experiences to be made, be it trying to survive or struggling to become rich. Navigate through the underground in search for valuable minerals and avoid running out of energy or health. Key features include random world generation, different upgrades for the player and a guild with daily random requests for resources.

Will you be the one to hit the mother lode and lead the high scores?

Gameplay description:

The game starts with a typical menu with options to choose from e.g. ‘‘Load Game’’, ‘‘Options’’ etc. After choosing a new or loading a saved game the user is presented with the start screen. This screen represents the city the player lives in and provides different choices. The user can visit the shop, a guild or enter an instance of ‘‘caves’’. Each instance is procedurally generated and thus random every time. The shop provides static improvements to the player attributes. The guild on the other hand requests specific resources and rewards the player upon return to the city after an instance if the minerals are mined and returned.

The main game starts with the created instance and focuses on the character upon entry. The mechanics are simplified for a prototype, the main user input consist of the arrow keys which let the player mine in the specified direction and the game camera always follows the player. The game world is restricted to the left, right and down so that the user has no way to go up again. To get out of the cave instance the player can use a ‘‘teleportation’’ item, which is available at the shop. Alternatively, the player returns to the city when either the health or the endurance is reduced to zero. Should the latter option come true then the player gets a penalty upon return to the city. During the instance the user just moves in the direction of the tile and triggers a mining animation. This action costs the player endurance and takes a specific amount of time depending on the current stats. Buying upgrades in the shop speeds up the action and reduces the amount of energy consumed. Furthermore, the player has a limited amount of space within his bag to hold the minerals, which can also be improved using the shop.

Should the player return to the city by having run out of health or energy then a penalty is added to the daily maintain costs, which include the costs for accommodation, tool repair and food. These costs remain the same throughout the game and can lead to poverty if the player does not collect enough resources. Collected resources can be sold to the shop at a fixed price.

The generation of the world/cave happens procedurally and depends on the player’s location in vertical direction. After a specific amount of movement downwards a new set of tiles is generated. Based on the current position and a random factor, various kinds of tiles with different mining difficulty will be instantiated. This results in a higher difficulty the deeper the player gets to. The game is not meant to end at a specific level, but this can be changed depending on the final playability. Mining Life is targeting the PC only but due to the simple input control this can be altered to any platform easily.

The user interface provides the option to see the current inventory and use the return item if it was bought from the shop prior to entering an instance.

Screen Images:

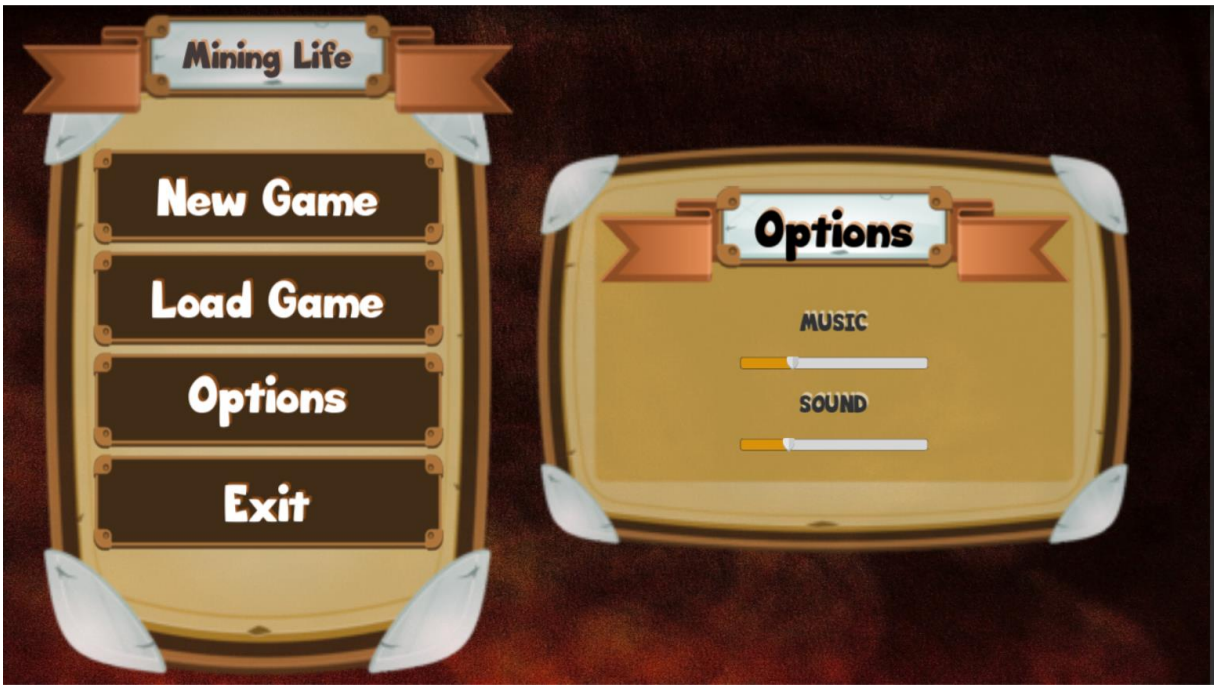

Figure 1 Main menu

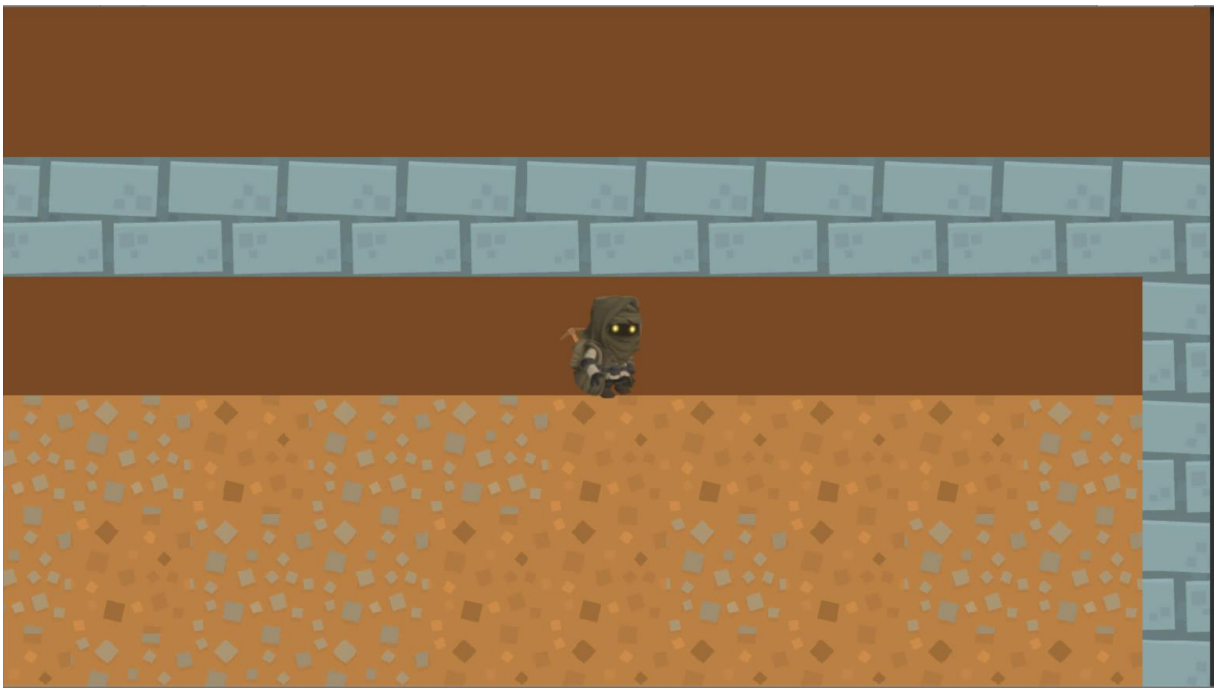

Figure 2 Cave instance

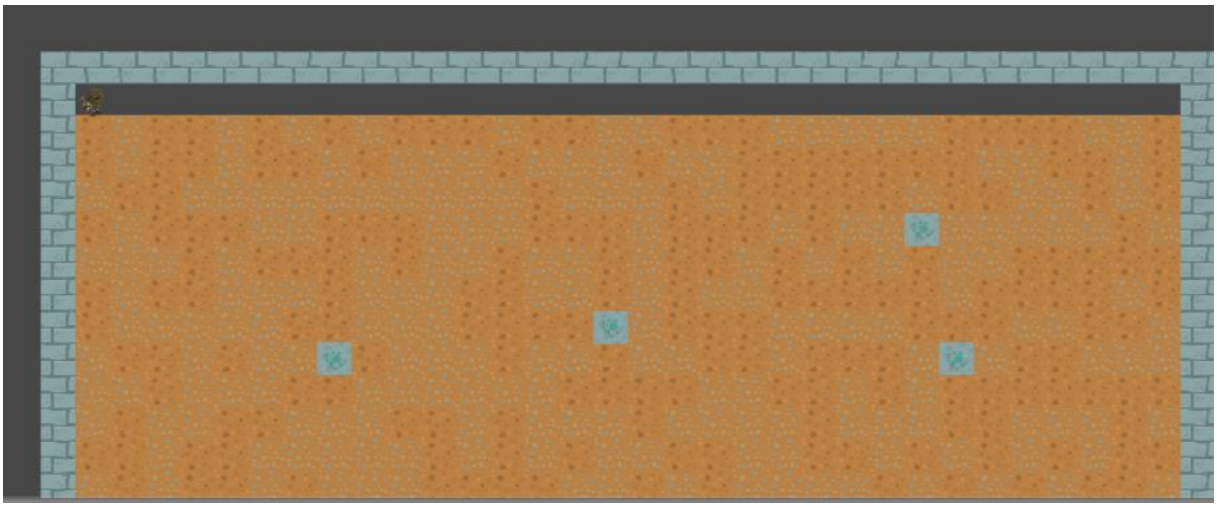

Figure 3 Instance from far away – currently only 4 tiles are in use

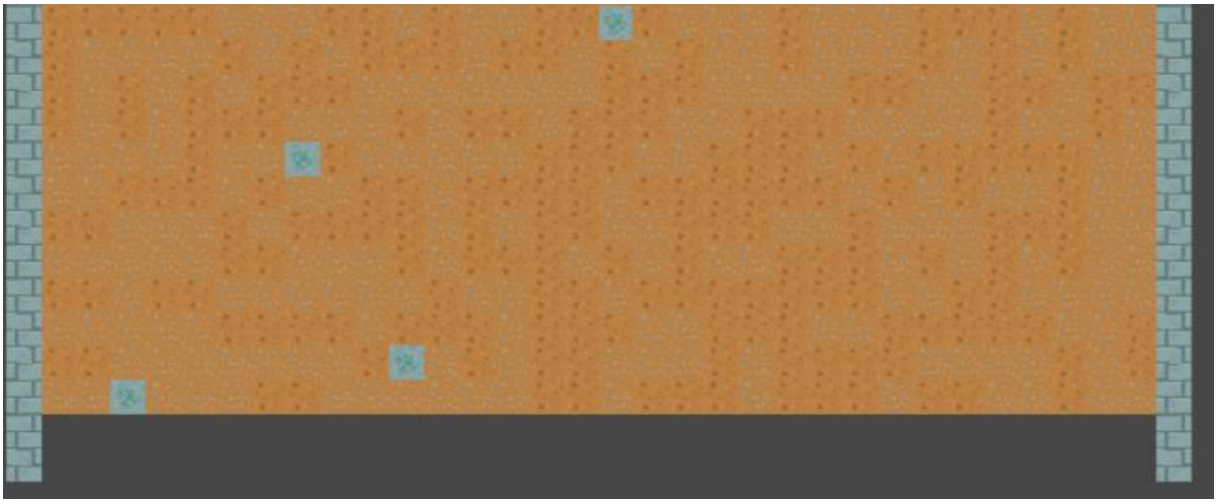

Figure 4 End of the instance - when player gets close then new terrain gets generated

Figure 5 Player mining downwards

Implementation:

The development environment chosen for this project is Unity3D. This game engine provides many useful features and enables developers to focus on the essentials of the game design instead of the usual concerns of complex tasks like math, science and physics which are required for any game engine that is written from scratch (Pereira, 2014). Mining Life is meant to be a two-dimensional game and Unity3D, as the name implies, has many more benefits for a three-dimensional game. Nonetheless, Pereira (2014) and Jackson (2014) state the many features and advantages this game engine has for example the physics, the sprite-workflow, animation, particle effects and much more.

One of the main technical challenges, besides the collision detection for the mining of a tile, is the procedural content generation. A Board Manager script handles all the level generation and almost everything is done on randomness. Sometimes too much randomness can lead to surprising results, but this can be prevented using simplified algorithms. As Watkins (2016) describes there is no real randomness in procedural generated content. This fact can be used for testing the game balance prior to publishing. For the cave instances a specific set of seeds can be chosen for the "Random" class and therefore be tested thoroughly. Especially, since Mining Life is a kind of infinite game this approach can be very useful. Another challenge is the correct collision detection since the tiles need to be distinguished correctly. To achieve this, a polygon trigger collider was added to the player character and in combination with the movement direction the correct tile is chosen.

```
public void Initialise ()
{
    //create a list to contain the vector3 positions of a grid
    InitialiseList ();
    //generate resources at random locations on the grid and remove available index from the list
    InstantiateAtRandom (resourceTiles, resourceCount.minimum, resourceCount.maximum);
    //Rest of the grid is being instantiated with tiles or outer walls respectively
    BoardSetup ();
    //instantiate player at the top
    Vector3 newLocation = new Vector3 (Random.Range (0, columns - 1), rows - 1, 0f);
    Instantiate (player, newLocation, Quaternion.identity);
    //find camera and position to player - set offset correctly
    mainCamera = GameObject.FindGameObjectWithTag ("MainCamera");
    newLocation.z = -10;
    mainCamera.transform.position = newLocation;
    cameraController = mainCamera.GetComponent<CameraController> ();
    cameraController.FindPlayer ();

}
```

Figure 6 Initialise method in Board Manager

Figure 6 shows the main workflow of the Board Manager. First, a grid is created, and each position is saved into a list by storing the corresponding Vector3 value. Secondly, minerals are generated on the grid by choosing random positions. These positions are removed from the list afterwards and the amount of resources that are generated is a random value between the given minimum and maximum count. The Board Setup method instantiates tiles for the rest of the grid. Afterwards, the player is positioned at the very top and the camera is scripted to follow the players location.

Furthermore, the difficulty of the levels and tiles are going to change based on the players movement downwards. After a specific depth a new level is generated with completely different tiles. This makes it inevitable to provide many kinds of sprites for the specific fields. In addition, all tiles all have unique attributes like health, mining difficulty and more. These attributes are responsible for the amount of energy the player uses upon mining the tile.

One of the key issues was the correct collision detection between the player character and the mineable tiles. Another aspect, was the fact that a normal box collider was not working properly for the player character since the movement stopped as soon as the slightest bumpiness appeared on the floor. To fix that, the player character needed a circle collider on the feet and a box collider surrounding the upper body. In addition, there is a polygon collider which forks into three different directions. The polygon collider is only a trigger collider and therefore is not considered for physical simulation.

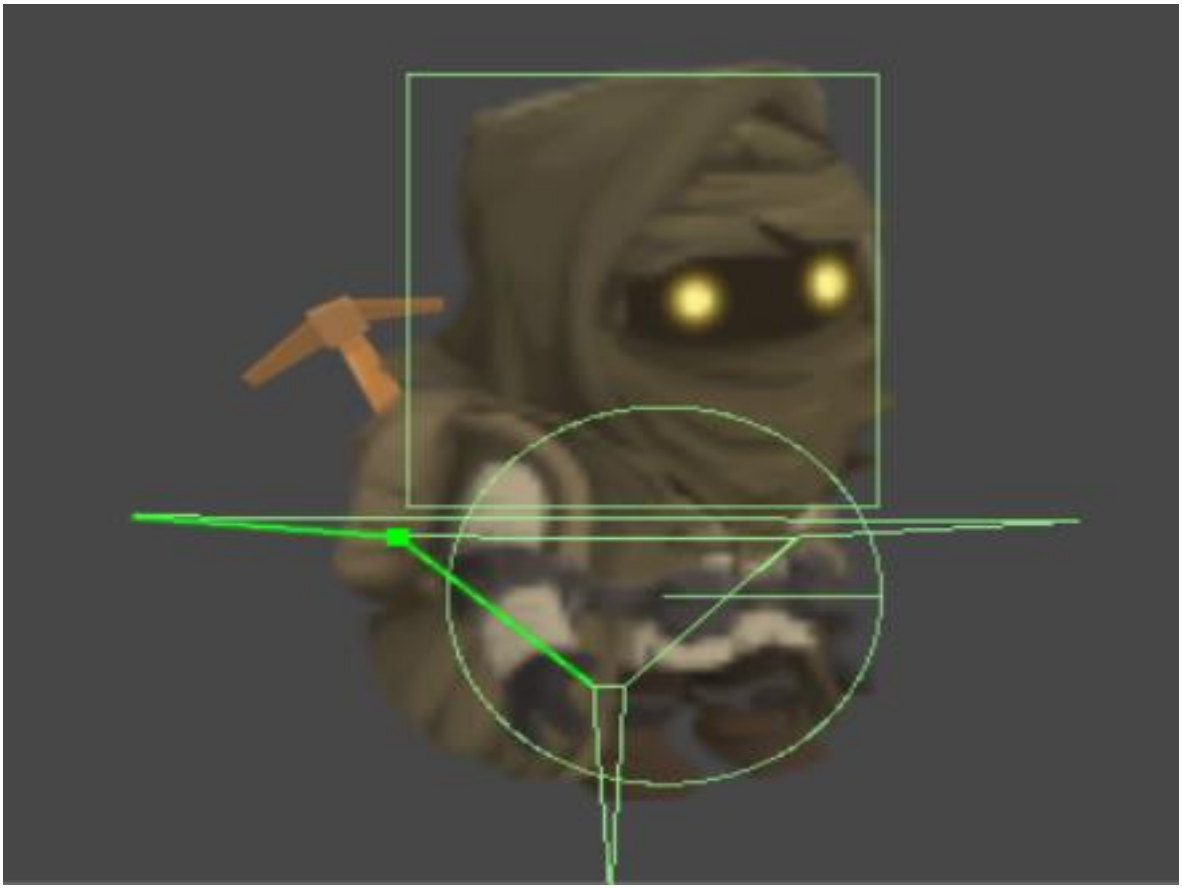

Figure 7 Player character with 3 colliders

The collision detection is done in Unity's "OnCollisionStay2D" method and particular care is taken to prevent adjoining tiles to be mined as well if they come in touch with the collider. This is done through disabling the movement in any other direction while the player is mining on a specific tile. Further implementation can be reviewed in the script "Player.cs".