Generic Blank Game Design Document (GDD)

Your Game Logo

Here

***‘The Bizarre Adventures of Lunk the Block’***

This page: Table of Contents and Team Member Listing

|  |  |  |
| --- | --- | --- |
| Table of Contents  [1 Executive](https://n-xovwktmtjsnaxyc2mwes2xu7pohqedmdm6zjw5q-2lu-script.googleusercontent.com/userCodeAppPanel) Summary  [1.1 Game](https://n-xovwktmtjsnaxyc2mwes2xu7pohqedmdm6zjw5q-2lu-script.googleusercontent.com/userCodeAppPanel) Overview  [1.2 Game](https://n-xovwktmtjsnaxyc2mwes2xu7pohqedmdm6zjw5q-2lu-script.googleusercontent.com/userCodeAppPanel) Summary  [2 Equipment](https://n-xovwktmtjsnaxyc2mwes2xu7pohqedmdm6zjw5q-2lu-script.googleusercontent.com/userCodeAppPanel)  [2.1 Hardware](https://n-xovwktmtjsnaxyc2mwes2xu7pohqedmdm6zjw5q-2lu-script.googleusercontent.com/userCodeAppPanel)  [2.2 Software](https://n-xovwktmtjsnaxyc2mwes2xu7pohqedmdm6zjw5q-2lu-script.googleusercontent.com/userCodeAppPanel)  [3](https://n-xovwktmtjsnaxyc2mwes2xu7pohqedmdm6zjw5q-2lu-script.googleusercontent.com/userCodeAppPanel) Evaluation  [3.1 Game Engine](https://n-xovwktmtjsnaxyc2mwes2xu7pohqedmdm6zjw5q-2lu-script.googleusercontent.com/userCodeAppPanel)  [3.2 Target Platform](https://n-xovwktmtjsnaxyc2mwes2xu7pohqedmdm6zjw5q-2lu-script.googleusercontent.com/userCodeAppPanel)  [4 File Formats & Naming Convention](https://n-xovwktmtjsnaxyc2mwes2xu7pohqedmdm6zjw5q-2lu-script.googleusercontent.com/userCodeAppPanel)  [5 Levels](https://n-xovwktmtjsnaxyc2mwes2xu7pohqedmdm6zjw5q-2lu-script.googleusercontent.com/userCodeAppPanel)  [5.1 Castle](https://n-xovwktmtjsnaxyc2mwes2xu7pohqedmdm6zjw5q-2lu-script.googleusercontent.com/userCodeAppPanel) & Dungeon Generation Code  [5.2 Asset](https://n-xovwktmtjsnaxyc2mwes2xu7pohqedmdm6zjw5q-2lu-script.googleusercontent.com/userCodeAppPanel) List     |  | | --- | |  | | Game Development Team Members    PRODUCER  Christian Barajas    PRODUCTION MANAGER  Christian Barajas    PRODUCTION COORDINATOR  Christian Barajas  GAME DESIGNERS  Christian Barajas  SYSTEMS/IT COORDINATOR  Christian Barajas  PROGRAMMERS  Christian Barajas  TECHNICAL ARTISTS  Christian Barajas  AUDIO ENGINEERS  Christian Barajas  UX TESTERS  Christian Barajas |

|  |
| --- |
|  |

# 1 Executive Summary

## Game Overview

Title: The Adventures of Lunk the Block

Platform: PC Standalone + iOS & Android

Genre: Procedurally Generated Dungeon Crawler

Rating: (10+) ESRB

Target: Casual gamer (aging from 12 - 30)

Release date: March, 2017

Publisher: C && C Corporation

Description: The Adventures of Lunk the Block is a 2D procedurally generated dungeon crawler. Players will play as the character “Lunk” where the user is in a castle attempting to explore the castle’s dungeons to find the ultimate treasures hidden in the dungeons of the castle. Players must survive using the three available weapons they are given which include a sword, bow and arrow, and lastly magic. Players must kill the castle’s enemies (slime-blobs, and man-eating snakes) as they make their way through different floors of dungeons. Along the way enemies will also drop gear and coins to further advance the player’s abilities/survivability. Players will also be able to level their character up as they acquire experience from killing enemies. In addition, there will also be shopping chests that will randomly spawn in the castle or dungeons for players to buy or sell items that may help them along the way. There are no allies, players will go in solo as a lone wolf to tackle the ultimate survivability test.

## Technical Summary

*The Bizarre Adventures of Lunk the Block* will be fully developed within the next several months beyond what was already developed. The plan is to continue to develop more of the game solo using the Unity game engine & some use of Adobe Photoshop. The total production cost of the game will exceed no more than USD $0. Because this is a for fun personal project, no revenue is expected. The game will be for PC and within the next several months I plan to hopefully have this game deployed for iOS and Android.

# 2 Equipment

## 2.1 Hardware

\*This project only used a Windows PC laptop in which I already owned.

*Product Task Cost Quantity*

Hp Pavilion Laptop Asset Creation $0 1

Game Development

**Total: $0**

## 2.2 Software

*Product Task Cost Quantity*

Unity Pro Game Editor/Engine $0 1

Adobe Photoshop (free version) Texture Painting

Photo Editing

**Total: $0**

# 3 Evaluation

## 3.1 Game Engine

The game engine utilized for the development of this game is Unity. Unity made the development of this game much smoother and it sped up the development process, optimize the game, and achieve a desired result without having to pull out too much hair.

## 3.2 Target Platform

This game is made for PC and I have a wish to make it compatible with smart phones but the main target platform for this game is for PC. On the other hand, it will be a great learning experience to tackle on the mobile development of this game. Learning mobile development would increase utilization by linking the project in two separate markets.

# 4 File Formats & Naming Conventions

*Asset Type SubType Naming Convention File Format*

Animations AnimationClipName FBX

Graphics Characters CharacterName PNG

Enviornment EnviornmentTileName PNG

PhotoShop

Props PropName PNG

PhotoShop

Materials MaterialName \*.mat

Scripts Character CharacterName C#

Enemy EnemyName C#

Attacks AttackType C#

Enviornemt EnviornmentName C#

Canvas CanvasType C#

UI UIType C#

UI UIElementState PNG

# 5 Levels

## 5.1 Castle & Dungeon Generation Code

levelCon.dungeonLevel++;

for (int i = 0; i < dungeonSize; i++)

{

for (int j = 0; j < dungeonSize; j++)

{

maze[i, j] = 1;

}

}

\*\* This code starts off the generateDungeon() method. The first line that increments dungeonLevel is provide which floor the player is currently on. For example when the game first starts dunegonLevel starts at 0, so when generateDungeon() is first called the screen states that the player is on floor 1. The nested for() loop is to make the entire castle floor or dungeon floor all bricks. Basically setting up the game floor. The line maze[i,j] = 1; maze[i,j] gets set equal to 1 because I associated the number 1 with wall tiles. For further reference: 0 = game floor

1 = wall tile

2 = floor exit

3 = torch object(left handed)

4 = torch object(right handed)

5 = spiked floor

maze[currentX, currentY] = 0;

for (int i = 0; i < iterationLoops; i++)

{

int tempRand = Random.Range(1, 5);

if (tempRand == 1)

{

if (currentY < dungeonSize -2)

{

currentY += 1;

}

}

else if (tempRand == 2)

{

if (currentX < dungeonSize -2)

{

currentX += 1;

}

}

else if (tempRand == 3)

{

if (currentY > 2)

{

currentY -= 1;

}

}

else if (tempRand == 4)

{

if (currentX > 2)

{

currentX -= 1;

}

}

maze[currentX, currentY] = 0;

if (i == iterationLoops - 1)

maze[currentX, currentY] = 2;

}

\*\* This set of code is to actually generate a game floor, either the castle floor or dungeon floors. The first line in this snippet, maze[currentX,currentY] = 0, sets the starting point for the floor. Then we use a for() loop to iterate through the entire game board. We use a random range between 1 and 5 to direct which direction the game board is being generated. tempRand of 1 goes upward, tempRand of 2 goes to the right, tempRand of 3 goes downward, and tempRand of 4 goes to the left. Once we have iterated through the entire for() loop we make another check. Once it is confirmed that we have generated the entire floor we set an exit for the level to continue onwards. This is the last if statement in this snippet. And again we set maze[current,currentY] equal to 2, because the number 2 is associated with the exit gameObject.

for (int i = 1; i < dungeonSize - 1; i++)

{

for (int j = 1; j < dungeonSize - 1; j++)

{

if (maze[i, j] == 1 && maze[i, j + 1] == 0)

{

if (Random.Range(1, 20) == 10)

maze[i, j + 1] = 3;

}

if (maze[i, j] == 1 && maze[i, j - 1] == 0)

{

if (Random.Range(1, 20) == 10)

maze[i, j - 1] = 4;

}

if (maze[i, j] == 0 && maze[i + 1, j] != 1 || maze[i - 1, j] != 1 || maze[i, j - 1] != 1 || maze[i, j + 1] != 1)

{

if (Random.Range(1, 100) == 1)

maze[i, j] = 5;

}

}

}

\*\* This code snippet is to fill the dungeon with either a torch gameObject or a spikedFloor gameObject. We use this nested for loop to iterate through the entire game board of the player is on. The first two if statements have two checks, the first part of the check is maze[i,j] == 1, this is to check if the tile we are currently on is a wall tile. Then for the first if statement the second check involves, maze[i,j+1] == 0, what this does is check if the tile to the right of the tile we are currently on is a floor tile, if both check happen to pass then there is a random chance for the tile we are currently on to have a right facing torch gameObject. The second if statement is the same thing as the first if statement but involving a left facing torch gameObject. Now the third if statement has several checks, the first check maze[i,j] == 0, is to check if the tile we are currently on is a floor tile, if this is true then we have 4 other check to make sure that any tile adjacent, in any direction, to the tile we are currently on is not a wall tile. We do these 4 other check so that when we generate our castle floor or dungeon we don’t generate any spiked floor gameObjects that would be unreachable to the player for their would be no point if the player cannot be hurt by them. Once all the checks pass then there is a random chance that the tile we are currently on will become a spiked floor gameObject.

for (int i = 0; i < dungeonSize; i++)

{

for (int j = 0; j < dungeonSize; j++)

{

if (maze[i, j] == 0)

{

if (levelCon.dungeonLevel <= 2)

Instantiate(floor1, new Vector3(j, i, 0), Quaternion.identity);

else if (levelCon.dungeonLevel > 2)

Instantiate(floor2, new Vector3(j, i, 0), Quaternion.identity);

if (i > dungeonSize / 2 && !spawnedShop)

{

Instantiate(shopChest, new Vector3(j, i, -1), Quaternion.identity);

spawnedShop = true;

continue;

}

\*\* The code snippet above is for actually spawning a level. The nested for() loops are to iterate through the game board tiles. The first check we make is, maze[i,j] == 0, for the floor tiles. As we loop through our game board we make a check to see what level the player is currently on, if (levelCon.dungeonLevel <=2), meaning that if we are on floors 1 or 2 which stand for the castle floors then we are going to instantiate floor1 tiles that are for our castle levels and not for our dungeons. Then the following, else if statement, checks to see if we are on floor levels 3 and above meaning that the player is exploring the dungeons instead of the castle, then we instantiate the floor2 tiles which correspond with the dungeon theme. Then after these checks, we have if(i > dungeonSize/2 && !spawnedShop). this line of code checks to see that if we have iterated through at least half of the level to iterate through and we have yet to spawn a shopping chest, then we need to instantiate one. Once we have spawned a shopping chest we set, spawnedChest = true, and then we use “continue” to continue through the following if statements as we iterate through the game floor.

if (levelCon.dungeonLevel <= 2)

{

if (Random.Range(1, 100) < 5)

{

levelCon.enemiesLeft += 1;

Instantiate(AI1, new Vector3(j, i, -1), Quaternion.identity);

}

}

else if (levelCon.dungeonLevel > 2 && levelCon.dungeonLevel <= 15)

{

if (Random.Range(1, 100) < 5)

{

levelCon.enemiesLeft += 1;

Instantiate(AI2, new Vector3(j, i, -1), Quaternion.identity);

}

}

}

\*\* After we have spawned our floor tiles and shopping chest for our game board, we now move on to spawning the enemies. So for the first if statement we check to see if we are in the castle levels, and if we are then there is a random chance at spawning a slime-blob enemy. If chances are that we do spawn one, then we must add 1 to the amount of enemies left on the floor and instantiate that enemy. The else if statement that follows checks to see if we are in the dungeon levels and does the same as the first if statement. Creates a random chance at spawning an enemy, but in the dungeons they are man-eating snakes instead of slime blobs. Then we add 1 to the amount of enemies that remain and instantiate them.

else if (maze[i, j] == 1)

{

if (levelCon.dungeonLevel <= 2)

Instantiate(wall1, new Vector3(j, i, 0), Quaternion.identity);

else if (levelCon.dungeonLevel > 2)

Instantiate(wall2, new Vector3(j, i, 0), Quaternion.identity);

}

else if (maze[i, j] == 2)

Instantiate(exit, new Vector3(j, i, 0), Quaternion.identity);

\*\* Now the code above deals with the instantiation of wall tiles and a floor exit to our level. So as we iterate through our game board, if (maze[i,j] == 1), meaning we are on a wall tile, we check to see what level we are on a castle level or a dungeon. If we are on levels 1 and 2, we are going to instantiate wall1 gameObjects which correspond to the castle levels. If we are on levels 3 and above then we are going to instantiate wall2 gameObjects which correspond to our dungeons. Now for that last else if statement, (maze[i,j] == 2), has to do with our level exit. If this is to be true then we are going to instantiate the exit gameObject to transition to the next level.

else if (maze[i, j] == 3)

{

if (levelCon.dungeonLevel <= 2)

Instantiate(torchLeft1, new Vector3(j, i, 0), Quaternion.identity);

else if (levelCon.dungeonLevel > 2)

Instantiate(torchLeft2, new Vector3(j, i, 0), Quaternion.identity);

}

else if (maze[i, j] == 4)

{

if (levelCon.dungeonLevel <= 2)

Instantiate(torchRight1, new Vector3(j, i, 0), Quaternion.identity);

else if (levelCon.dungeonLevel > 2)

Instantiate(torchRight2, new Vector3(j, i, 0), Quaternion.identity);

}

\*\* The code above deals with the instantiation of our torch gameObjects. If, (maze[i,j] == 3), then we check to see if we are on a castle level or a dungeon level and we then instantiate the corresponding torchLeft gameObjects. torchLeft1 for castle, and torchLeft2 for our dungeons. Then we repeat the exact same code but for our torchRight gameObjects, which correspond with, (maze[i,j] == 4).

else if (maze[i, j] == 5)

{

if (levelCon.dungeonLevel <= 2)

Instantiate(spikedFloor1, new Vector3(j, i, 0), Quaternion.identity);

else if (levelCon.dungeonLevel > 2)

Instantiate(spikedFloor2, new Vector3(j, i, 0), Quaternion.identity);

}

}

}

\*\* Now for this last bit of code. This last check is for the instantiation of our spikedFloor gameObject. If, (maze[i,j] == 5), then we are going to instantiate the appropriate spikedFloor. If we are levels 1 and 2 we instantiate spikedFloor1 tiles to represent the castle floor spikes. And if we are on levels 3 and above then we instantiate spikedFloor2 tiles to represent the dungeon floor spikes.

## 5.2 Asset List

Players Lunk the Block

Enemies Slime Blob

Man-Eating Snake

Props torchLeft1

torchLeft2

torchRight1

torchRight2

shopChest

Enviornment spikedFloor1

spikedFloor2

floor1

floor2

wall1

wall2

exit

# 