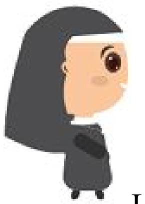


50.003
[Elements of Software Construction]
PROJECT REPORT

Lim Ying Jie | Paul Timothy Tan Bee Xian | Seng Jia Hong | Tan Li Xuan

SYSTEM REQUIREMENTS

Game Features

1. **Cross Platform**

Nun Story can be played on several platforms such as Android, IOS, Windows and Mac, using the build targeted at each respective platform, of course. For example, players using Android and Windows are able to play together in the same game instance. This allows the users to play the game everywhere and anytime on any devices as long as they are connected to the internet.

2. **Multiplayer**

The game can be played with 2 to 4 players at one time. A single player initiates the game by forming a new group which is able to support up to 4 players. This allows friends to compete against one another in a friendly setting.

3. **Parallax Scrolling Effect**

Parallax scrolling effect creates an illusion of 3D depth in a 2D scene and provides an immersive experience for the player as he progresses through the game. The game object containing the background tracks the movements of the local player, and adjusts its transform position accordingly, such that it moves a fraction of the distance travelled by the player in the horizontal axis. From the perspective of the camera attached to the player, as the player moves to the right, the foreground objects and the background image appear to scroll to the left, but in reality, the background moves at a slower rate, thus achieving the effect of having the background “appearing” to be in the far distance.

4. **Tiled Sprites**

The usage of tiled sprites was a design choice we made as it offers us some benefits during the development process. The modularity of the tile set allows the level designer to extend the map without the need for additional sprite assets to be drawn. Furthermore, it is easy to extend the map via an implementation of a procedural map generation algorithm. This cuts down on development time and file sizes.

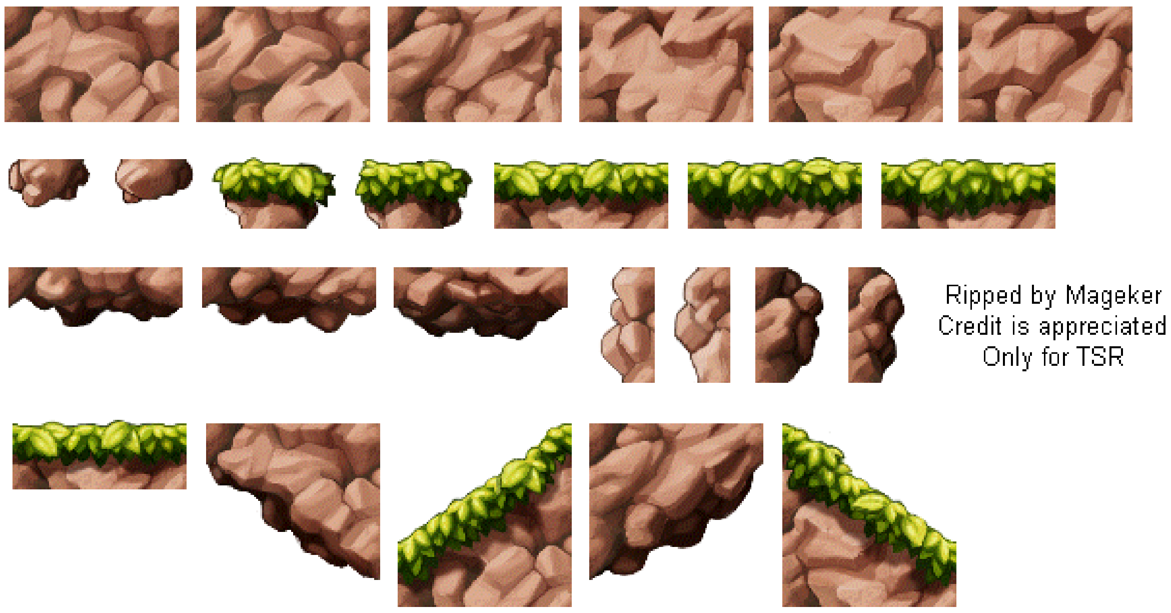


Figure: 1: Tiled Set obtained online. Used for the creation of the map in our game.

User Interaction

1. **Random Generated Items**

There are multiple items in the game which are obtained via pick-ups from the "Question Mark Boxes". The items are randomly generated and the pick up boxes would reappear after a certain time has passed. Each player has an equal chance of obtaining all items. Each item has its own unique feature but all the items serve to impede other players' progress in the game. Certain items has an attack range which may affect the local player as well if he is within the attack range. Each player is allowed to hold one item at a time only.

Figure 2 : The "Question Mark Box" used in the game. Player passes through the box to obtain an item.

2. **Interactive Map**
   a. **Multiple Paths**

The map is interactive and consist of several paths, traps and movable platforms. Players are provided with multiple options when navigating through the entire course. Figure 1 below shows how certain routes are more advantageous and contains more "Question Mark Boxes". However, they require more skill and effort to reach it. This provides an incentive for players to attempt acquiring the items, as the items would prove to be advantageous in slowing down their opponents.

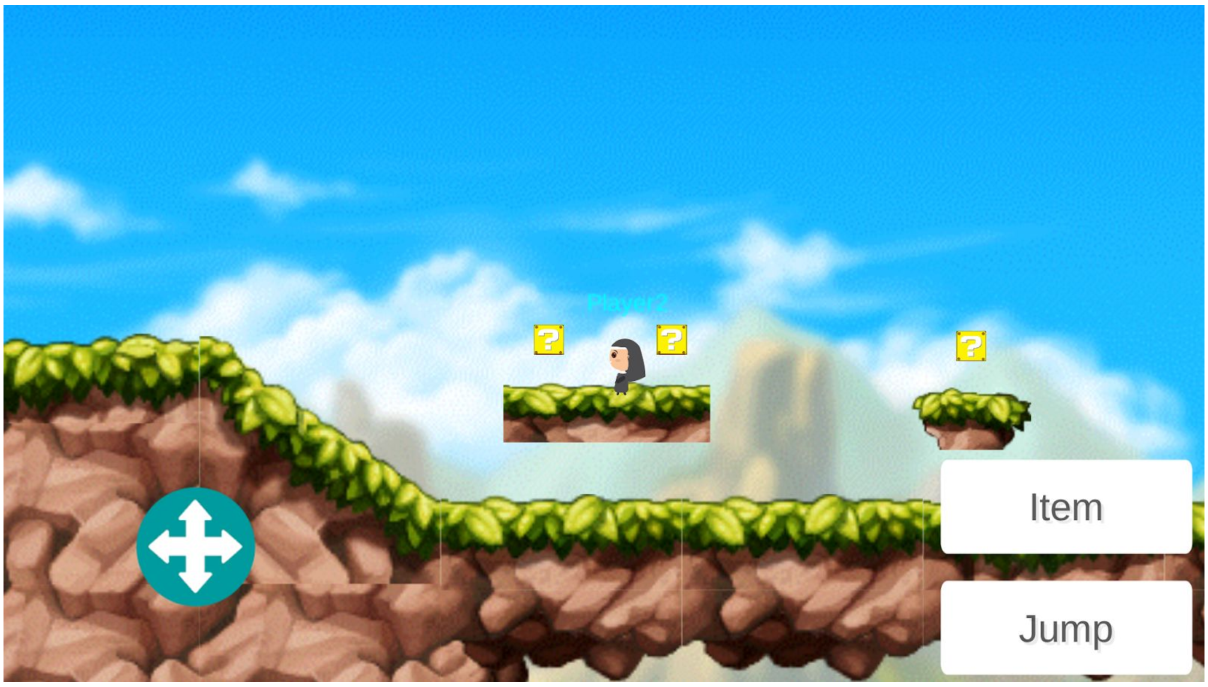


Figure 1: The player has to jump onto the platform to be able to obtain items from the “Question Mark Boxes”.

b. Landmines

There are also traps in the course such as landmines which the players must jump to avoid it. Should a player walks into a landmine, he would be blown back. To make it easier for the players lagging behind the first player, as soon as the landmine is detonated, it would not reappear. Therefore, this makes it harder for the first player to proceed quickly as he has to proceed with care while the players behind has a better chance in closing the gap. This provides a balance in the game and prevents the last player from falling way too far back.

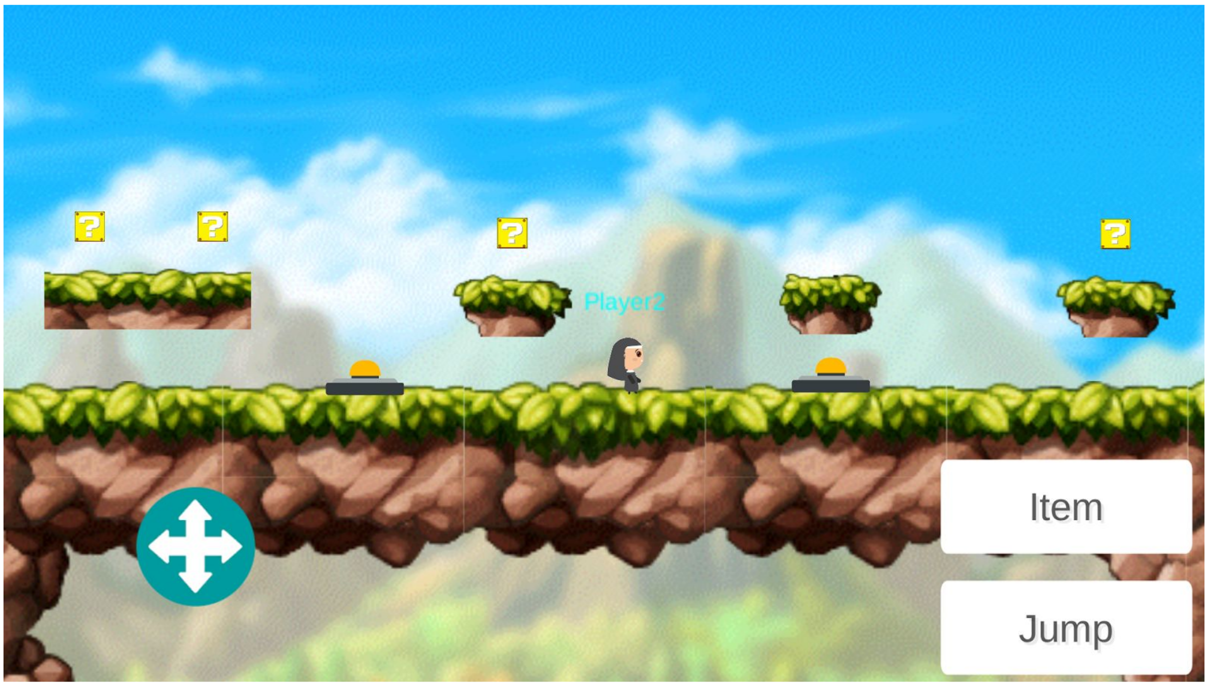


Figure 2: Player 2 has to jump onto the platform to prevent stepping onto the landmine.

c. Movable Platforms

Moveable platforms in the map serve to facilitate the movement of the players as they are unable to jump high enough to reach the top of the cliff. The platforms are a non-predictability factor in the map as they are not static and changes location as the time of arrival differ. Players are forced to share the platform which in turn creates opportunities for player-player interactions through item usage. Although fun, the implementation of the movable platform in this scenario was not ideal due to game balance. Steps for improvement is noted in the later section of this report.

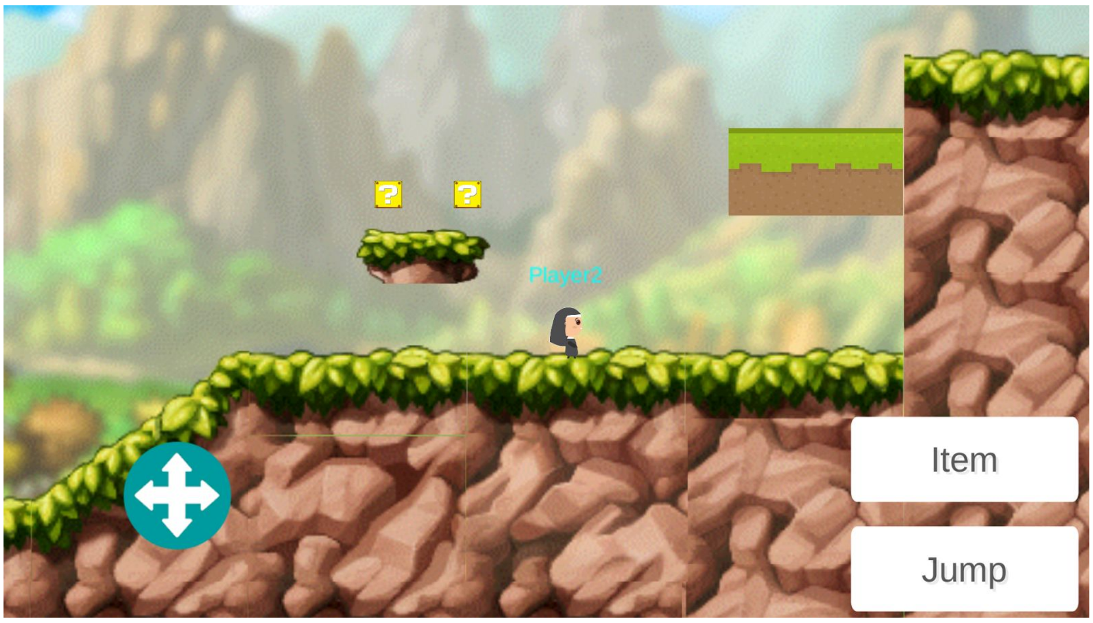


Figure 3: Player 2 has to wait for the moving platform to descend and ascend before he is able to reach the top of the cliff.

3. Status Effects

We want players to be able to interact with each other through the pickups system, we would allow them to use the items to impede the other players' progress. These status effects (commonly referred to as Crowd Control in other games), when inflicted on other players, will impair their ability to move.

SYSTEM DESIGN

System Implementation Components

1. GameObject

In Unity, all things are based around the GameObject and components. The GameObject is a container in which we add various components, to give it various properties which make it into whatever we want it to be. For example, a camera GameObject would have a Camera component attached to it, while a bullet GameObject would have a Sprite Renderer, Collider 2D, Rigidbody 2D components as well as various scripts attached to it.

Scripts are created and attached to a GameObject to give it custom behaviors. These scripts inherit from Unity's builtin Monobehavior class which have functions that are called at the start of Game Object activation and during every event loop.

2. **The Scene**

The Scene contains the GameObjects for our game. It is used to compartmentalize our game into different parts, such as the Lobby, and the Main game scene itself. It is in the Scene View that we build our game world, by positioning various GameObjects such as map tiles, obstacles and decorations.

3. **Unity Collab**

Our project was coordinated using Unity Collab. We practiced publishing frequently and avoiding working on the same scripts, scenes and game objects to minimise errors or loss of progress.

Communication Between Components

When coding, our scripts need to communicate with each other, need references to items in the game and need to pass data around with efficiency and speed. This is done primarily via the GetComponent<>() method.

1. **Accessing other components attached to the same game object**

```
GetComponent<Transform>().Translate(0, 1, 0);
```

2. **Accessing methods in other scripts attached to the same game object**

```
OtherScript otherScript = GetComponent<OtherScript>();
otherScript.DoSomething();
```

3. **Accessing methods in a script attached to a different game object**

```
void OnTriggerStay(Collider other) {
       if (other.GetComponent<OtherScript>())
           other.GetComponent<OtherScript>().DoSomething();
}
```

Instantiating a new GameObject from a prefab and then calling methods in a component attached to the new GameObject

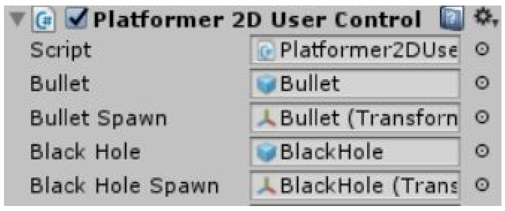
```
Instantiate(fragment, new Vector2(transform.position.x + 2 * Mathf.Sin((i
* Mathf.PI) / 180f), transform.position.y + 2 * Mathf.Cos((i * Mathf.PI) /
180f)), Quaternion.identity).GetComponent<Rigidbody2D>().AddForce(new
Vector2(50 * Mathf.Sin((i * Mathf.PI) / 180f), 50 * Mathf.Cos((i *
Mathf.PI) / 180f)));
```

CODE STRUCTURE

Game Objects
The game objects we used in our game are instantiated from various prefabs which we created. Each prefab has its own script attached to it, which controls how the game objects should behave in the game.

Variable Accessibility
It is good practice to restrict the accessibility of variables and methods to `private` if it is not meant to be accessed from outside of the class, as this provides encapsulation, which helps to prevent bugs resulting from unintended access to the variables from other scripts.

The variables we have in our game's code can be categorized into three accessibility types:

1. **Public variables whose values are meant to be modified from the Unity Editor**
```
public GameObject blackHole;
public Transform blackHoleSpawn;
```


Platformer 2D User Control

Script	Platformer2DUse
Bullet	Bullet
Bullet Spawn	Bullet (Transforn
Black Hole	BlackHole
Black Hole Spawn	BlackHole (Trans

2. **Private variables which can only be accessed from within the same script**
```
private float duration = 5f;
```

3. **Private variables with public methods implemented to allow other scripts to access and modify its values**
```
private bool isStunned = false;
private float stunDuration = 0f;

public void applyStun(float duration)
{
    isStunned = true;
    stunDuration = duration;
    print("tio stun liao");
}
```

Interaction between Scripts
The Platformer2DUserControl script is responsible for handling the player's actions, such as movement and usage of items picked up. It serves as a central nexus with which the other scripts for the gameplay related items, such as the black hole and the rockets, interact with.

The scrolling background is made possible by a Background script, which attaches itself to the main camera and adjusts the backgrounds position based on camera's movement. The CameraFollow script on the main camera then attaches to the local player and follows them.

Since the background movement is handled locally, this allows each player to have a background follow them, independent of the other networked GameObjects.

Comments
The scripts have been annotated with comments detailing at a high level what the code does.

<u>**TESTING**</u>

Black Box Testing

Black box testing is done primarily to verify the functionality of the visual elements of the game, such as the map, sprites & animation, UI and gameplay.

Networking:

1. Loaded the server by spawning a very large number of items, from each device. Game will not crash, but network transfer rate will increase and load on local phone will increase. This will make the game have slower refresh rate, but is still playable.
2. Checked for accurate positioning of non-local players. Played game with all members side by side and multiple moving platforms. Watched out for unnecessary jerkiness (during early movable platforms) and non-updating player positions (bug is stassis code, applying constraints locally). Movement delay is reliant on network ping.
3. Checked for accurate trajectory of non-local player spawn items. Created multiple boxes that would interact with objects locally. Tested each item and compared the behaviour of the locally handled box-behaviour. Results were mostly the same, little discrepancies due to random collision of boxes locally.
4. Checked for synchronized animations across devices. Simply played next to each other and monitor animations for each interactive object.
5. Playing on PC tended to have slightly worse networking performance in terms of updating position of objects and players. Build is optimised for android devices. PC not recommended for competitive play.

White Box Testing

White Box Testing is used to check that all code branches are being executed properly as intended. This is done with the help of debug messages being printed to the console, which visually allows us to determine which code branch is currently being executed.

Regression Testing

After the iterative implementation of each new feature, we conduct regression testing to ensure that we did not introduce any new bugs into the game. It is carried out using both of the above testing methods.

CONCURRENCY

Our project is networked using Unity's provided network manage and multiple custom scripts for our networked objects implemented using [Command] and [ClientRPC]. To ensure an enjoyable player experience, we followed certain concurrency goals when implementing the system.

Goals:

1. Minimise data transfer by sending only essential data. Keep physics-based calculations local as much as possible. This ensures network smoothness.
2. Uniformity in client experience, either through server reference or reliable prediction. This ensures predictable and fair gameplay.
3. Safety nets in the case of unexpected behaviour, by updating essential data periodically when needed. This improves the robustness of the system for a better player experience in sub-optimal conditions.

Player position and animation

Player position is managed using network manager and network transform.

The NetworkTransform component synchronizes movement of game objects across the network. This component takes authority into account, so LocalPlayer objects (which have local authority) synchronize their position from the client to server, then out to other clients. Other objects (with server authority) synchronize their position from the server to clients.

However, although the sprites of non-local players were moving, animations were not playing correctly. The non-local player sprites would "slide" around the map.

To fix this, we passed command variables: jump and move, to the server. The local-player would then receive the variables from all other non-local players, and execute the move command locally which is tied to the animation. Thus, the correct animation for non-local players is shown based on the commands of the other players, while the movement is taken care of by network transform.

Interactive map objects

1. **Movable platforms**

Initially, we created movable platforms based on the formula:

```
transform.Translate(new Vector3(0, Mathf.Sin(Time.time), 0) * magnitude *
Time.deltaTime);
```

This formula would cause the platform to move sinusoidally due to the Sin function with change in time as the input. This gave us the desired result in single player mode.

However, in multiplayer mode, the formula would take the `Time.time` of the local player. Thus, the host and clients could have platforms that were out of sync due to different start time. This would result in non-local players being transported by “invisible” platforms from the perspective of local players.

The network manager also did not support the spawning of the platforms on start. Simply changing the game objects of the platforms to NetworkBehaviour and attaching network transform would only partially fix the problem. Although period and phase would be the same, they were instantiated at different positions, thus some platforms would be higher than others. This was due to all clients creating the platforms locally, instead of on the server. Thus, there would be N copies of the same platform, instantiated based on when the local player started the game.

This was finally fixed by implementing a platformSpawner, which created all instances of the platform in the server, when the server is started. Thus, only one platform is created on the server and all clients reference that. Networktransform updates were also changed to 0 as the movement could be calculate locally!

2. Landmines

Similarly, the Landmines were all generated locally during single player mode. In multiplayer mode, there would be a copy of each landmine being spawn locally. This was initially not a problem, as the network manager would synchronize the movements of the players on all devices. Since setting off a mine is entirely position related, the network manager could successfully simulate the non-local players running into the mine and detonating each mine locally.

However, as more players joined and more items were created, this solution did not behave reliably. Because of packet loss from poor connection or too many items spawning, the network transform would not accurately send the player position to other devices. This caused a frequent problem of local players touching the landmine and being pushed far away almost immediately. But the server would never show other devices that the player touched the landmine as they only collide for a few milliseconds before being push away, shorter than the maximum refresh rate of the network transform (29hz). Thus, the landmine would not be reliably detonated on every device. Hence, on collision, the landmine would be detonated locally, but still exists on every other non-local device.

This was solved again by creating a landmine spawner which would create all landmines on the server when the server starts up. Thus, when a client walks into a landmine, he would destroy it on the server and all other devices would have the landmine destroyed as well.

However, this introduced a few problems as the animation of the landmine was not being played properly when non-local players detonated them. This is due to the landmine not being immediately destroy on collision, but stays alive for 0.75s to play the detonation animation, before being destroyed on the server.

This was solved using [ClientRPC] commands to tell all other devices that the landmine was setoff and destroying on the server only after the animation was played locally. Thus, when a landmine was detonated locally, other devices would immediately play the animation and detonate after finishing.

Item spawning and pickups

Item spawning was handled by the network manager, and movement of the item generated was handled locally.

Whenever the player uses an item, he spawn the item object on the server with a certain velocity. The movement is handled locally as the same physics apply on all devices. Thus, there will be very little discrepancy between devices regarding item movement behaviour. Handling movement locally also greatly reduces data being sent of item movement, decreasing server load.

Pickups were also handled locally as collision is completely movement based, which is already handled by network transform. Unlike the landmine, where the player only collides for very short few millisecond window, the collision with the item pickup box is very reliable as there are rarely any sudden movements during collision.

Status Effects
When the player is affected by a status effect, he sends his status to the server, which sets the status of the player object on other devices to the new status effect. This is done by invoking the [command] function.

However, since status effects are often accompanied by large numbers of items spawning, packet loss may occur and interfere with setting the status on non-local devices. For example, when a player is stunned, his status is set to stunned and he sends his stunned status to the server. After his status has reverted back to normal, he tells the server again to turn his status back to normal. However, sometimes the status will be “stuck” as the messages may fail to send/update due to packet loss.

Thus, a repeating status notification was programmed to update the status of each player at a regular interval only if the local status is set to normal.

IMPROVEMENTS

Movable platforms

The scenario shown in Figure 3 was not ideal since it does not enhance the game balance. There are two possible scenarios which may occur. Assuming player 1 is in the lead and player 2 is lagging behind:

Scenario 1: Player 1 gets on the movable platform first and proceeds onwards. Player 2 has to wait for the movable platform to descend and ascend. In this scenario, Player 2 does not benefit at all and in fact, Player 1 is further in the lead. This does not encourage player to player interaction.

Scenario 2: Player 1 arrives first but has to wait for the movable platform. Thereafter, Player 2 arrives. Both arrives the top at the same time. This scenario encourages player to player interaction.

Firstly, we could improve on the movable platform by implementing several different path options at the same location. Players could either chose to navigate through the course with skill or opt to use the moveable platform which would take a longer time. Therefore, players who are more skilled would benefit from this arrangement.

Secondly, the movement of the platform could be expanded to multiple platforms in a circular motion or horizontal motion. Players would be tested on their jumping skills which slows them down. Furthermore, the players ahead has a higher chance of getting hit from the back as they focus on navigating through the course.

INSTALLATION AND USER MANUAL

Installation

Connect your device to the computer. On your phone, go to *Settings* > *About phone* > and tap the *MIUI version* 7 times continuously. Then go back to *Settings* > *Additional settings* > *Developer options* > and check *USB debugging*. Now scroll down and find *Select USB Configuration*. Tap and select *MTP (Media Transfer Protocol)*. Now you must make sure that third-party apps are allowed on your phone. Go back to *Additional settings* > *Privacy* > and check *Unknown sources* to allow installation of apps from sources other than Google Play Store. Now, on the computer, go to the phone folder and copy the APK file into the "Internal Storage" folder. Now, on your phone, go to *Explorer* > *APKs* > *NunStory.apk* > and tap *Install*. Congratulations, you now have NunStory installed on your phone!

User Manual

Nun Story is a multiplayer running game between 2 - 4 players, who use pick ups to try and finish in first place.

1. **Lobby page**

You could choose to just host a game (and not play), create a game and wait for other players to join, or join a game hosted by another player.

Hosting a game

If you will just be hosting the game and not playing, press the "Dedicated Server" button; if you will be playing the game, you can either use the Matchmaker and create a game that will be listed in "List Servers" (other players can find and connect to your game via "List Servers"), or you could host a game which is not listed (other players need to manually connect to the game using your IP address).

Joining a game

You can either use the Matchmaker and join another player's game via "List Servers", or manually connect to a game by typing the host's IP address.

Once there are two or more players in the lobby, when all the players have pressed "Join", there will be a short countdown after which the match will start.

2. **Playing the game**

Once the race starts, there will be three buttons. One of these is the movement joystick, on the bottom left-hand corner of the screen. The other two buttons are the jump button and the item button, which are on the bottom right-hand corner of the screen.

Run through any of the question mark boxes in order to pick up an item (which might enable you to impede another player's movement). There are four items.

The black hole will pull opponents within the range towards it and hold them there until the black hole expires, thus impeding their movement.

The holy hand grenade and the rocket will stun opponents within the explosion range, thus impeding their movement.

The “bun-munition” will slow down whichever opponent it hits. “Status effects” will be sent to whichever opponent was hit.

There are also landmines which will cause you to bounce off if you step onto them. Jump to avoid them.

The winner is the one who finishes the race in first place. Enjoy the game!