MapReduce

Team member

Xinyu He – xh3775 – hxy1303@gmail.com

Rongkun Wang – rw28478 – wangrongkun@utexas.edu

Introduction

MapReduce (the same name as the big data platform in Google) is a rogue-like video game. As a player, you need to beat three stages to gain the victory, and the difficulty increases as you progress further.

Dependencies

It does not have any third-party dependencies. It does not require any permissions either.

Build

It is a standard Android application. We develop it on Pixel 2 API 28 AVD, but it should work on almost any devices.

Game flow

There are three activities in the App: a starting activity, a game activity and an ending one. We spent most of our time in the game activity.

A player starts the App and reaches the starting screen. He/She can choose between starting a new game and read the manual.

There are three stages in every game. Once a player beats the current stage, he/she will automatically advance to the next one. The stages get harder by increasing stats that the enemies on the way. The player becomes stronger by collecting items, awards, and buy stuff in the shop. Once a player wins or loses a game, he/she will receive an ending page, and may quit the App or restart a new game.

Due to the nature of rogue-like games, each game is randomly generated that the player will get an unique gameplay experience every time. Also, when the player dies, he/she will lose all the previous progress. Although the rules may seem to be painful at start, they help make the gameplay more immersive. Players will have to think carefully about their decisions. Every failed run provides valuable information about the game, which is essential to beat the game.

UI

There are plenty of information in the main game view:

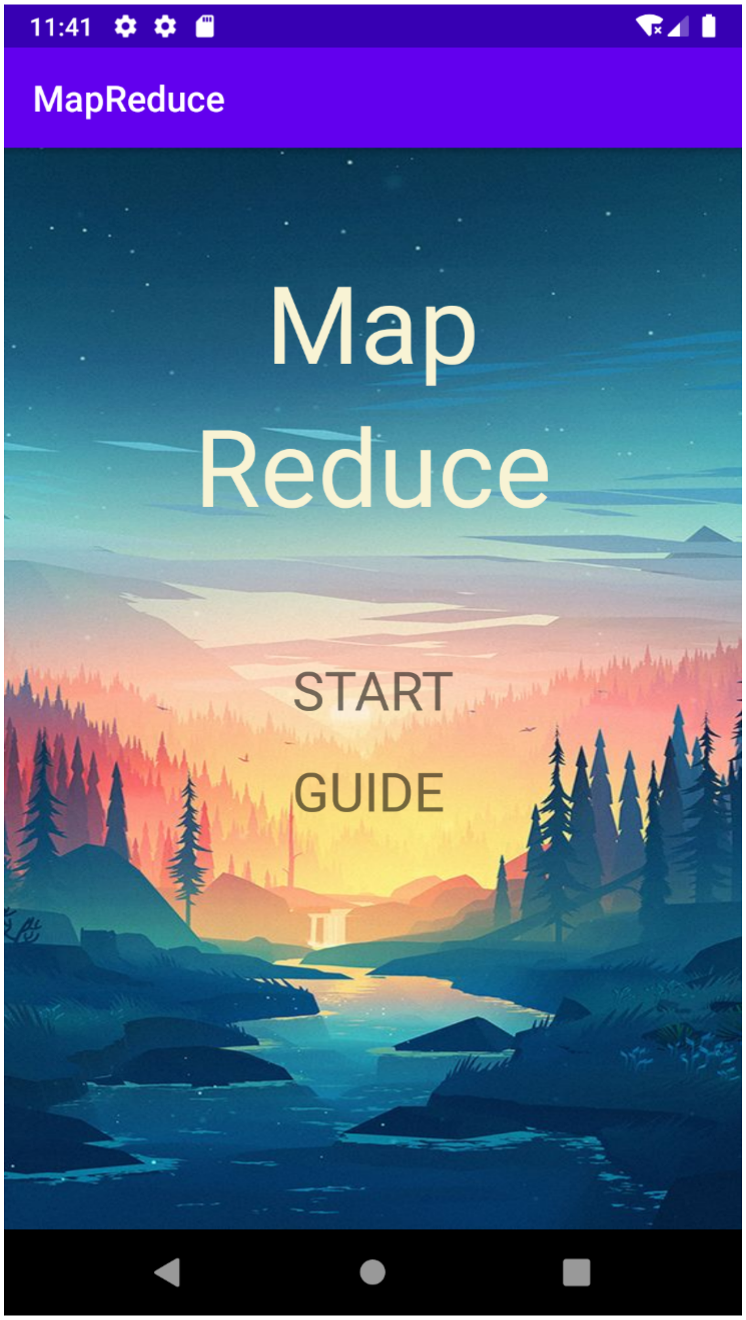


Figure 1: starting screen

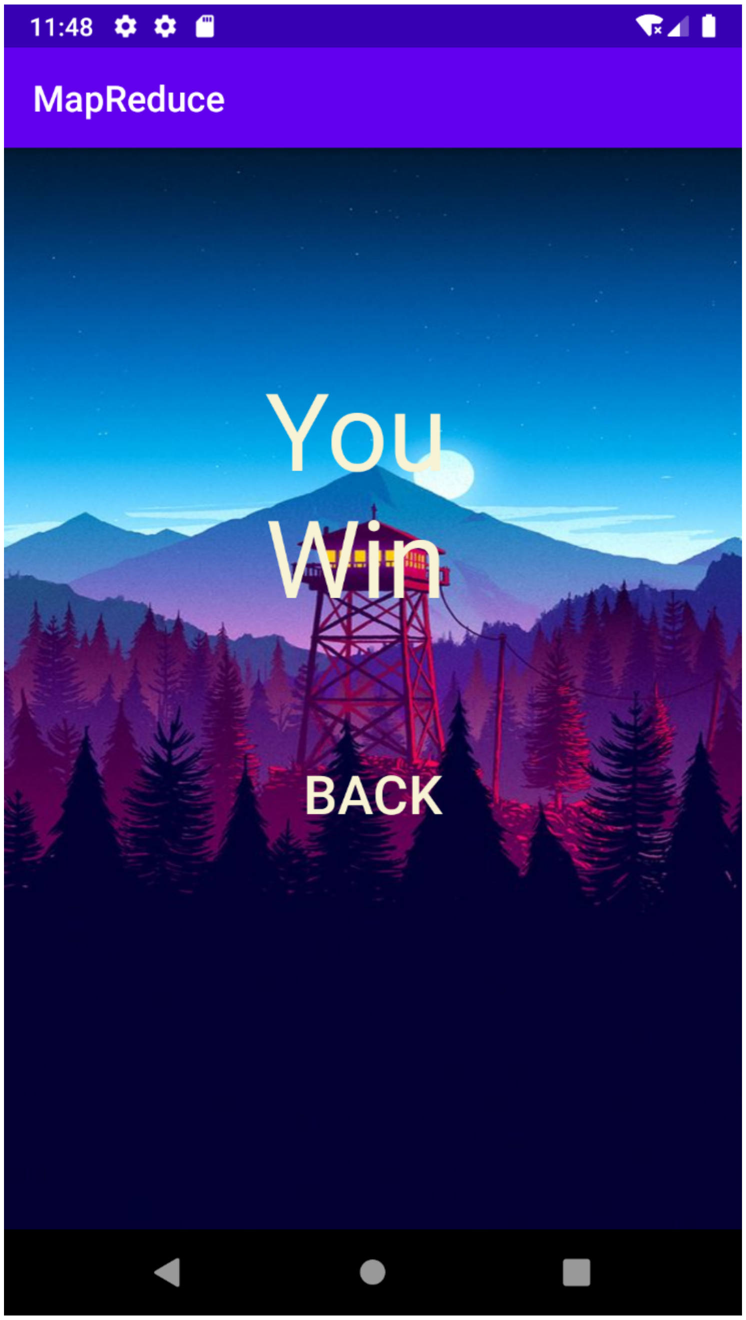
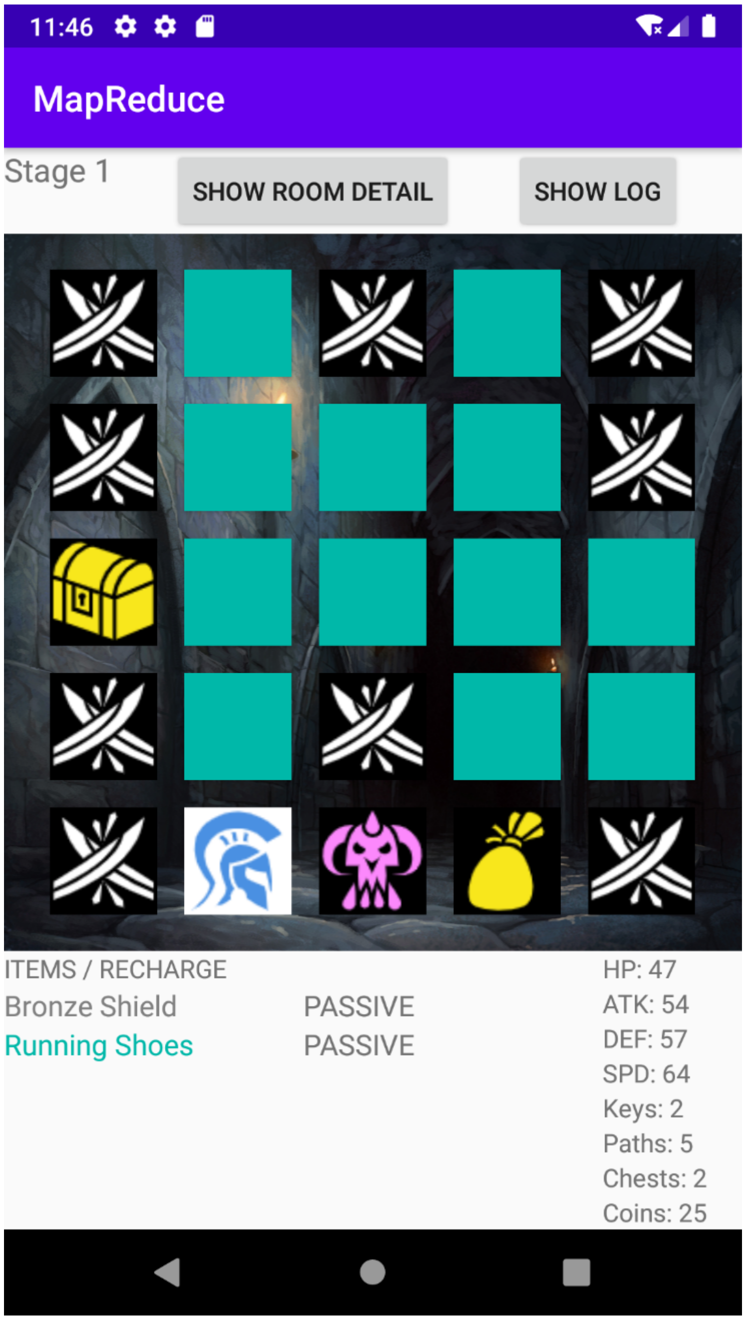


Figure 2: ending screen



Stage 1

ITEMS / RECHARGE

Bronze Shield	PASSIVE
Running Shoes	PASSIVE

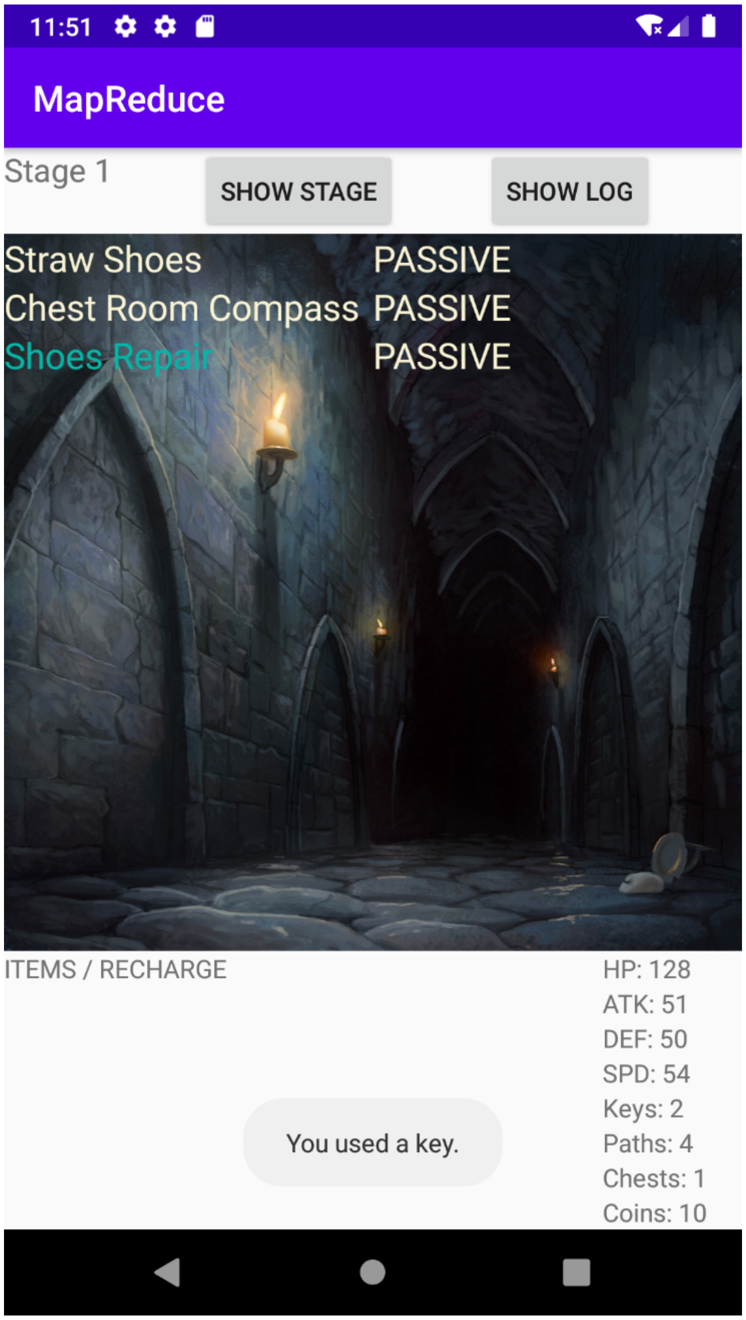
HP: 47
ATK: 54
DEF: 57
SPD: 64
Keys: 2
Paths: 5
Chests: 2
Coins: 25

Figure 3: main game

- On the top left corner, it displays the current stage.
- On the top middle, a switch button enables the player to switch between the main game view and the room detail view.
- On the top right corner, there is a button for showing/hiding the game log. The log provides a durable way for the player to review what happened in the game.
- Every stage consists of 5 * 5 = 25 rooms. The room can be one of the four kinds: normal room, boss room, chest room and shop. Once a room is visited by the player, he/she cannot visit it again, but can still go across it to enter further rooms. The icon on the room shows its kind. For rooms without icons, they are either visited or cannot be reached yet. The player location is shown by the helmet icon.
- Although not explicitly shown, there may not be paths connecting two adjacent rooms. Player must build a `path` in order to enter. In fact, the paths are the scarcest resource in the late game.
- On the bottom left corner, there is a `recyclerview` showing the items obtained by the player. Items are either `passive` or `activated`. The `activated` items need to be fully recharged for use, while the `passive` ones grant permanent effects.
- On the bottom right corner, it displays the current statistics of the player. Detailed description:
  - HP: the amount of damage the player can take before death.
  - ATK(Attack): the amount of damage the player can deal each attack.
  - DEF(Defence): the amount of damage the player can block before losing hp.
  - SPD(Speed): it decides who attacks first.
  - `Keys` are used to open chests, chest rooms and shops.
  - `Paths` are used to build paths between two adjacent rooms.
  - `Chests` contain some random award, and require a key to open.
  - `Coins` are used to buy items in shops.

When the player clicks a room, he/she will be taken into a room detail view. A room detail view is essentially a `recyclerview`.

- For normal rooms and boss rooms, it lists the enemies that the player MUST beat to advance. To make things simple, the battle is automatic, and the player can view the full log later.
- For chest rooms, it lists the `items` that the player can choose to take. A player can only take one item from a chest room. A player can exit a chest room without taking any items by pressing the back button.
- For shops, it lists the shop items that a player can spend money buying. A player can exit a shop by pressing the back button.



Stage 1

Straw Shoes	PASSIVE
Chest Room Compass	PASSIVE
Shoes Repair	PASSIVE

HP: 128
ATK: 51
DEF: 50
SPD: 54
Keys: 2
Paths: 4
Chests: 1
Coins: 10

You used a key.

Figure 4: room detail

Great Features and Challenges

UI

- The room status in a stage changes over-time. Hence we need to redraw the stage to reflect this. All redraw logic happens in `redrawStage`, where everything happens dynamically. A `HashMap` handles the mapping of view ids to room ids, allowing us to get the room object when the player clicked a room. The most hacky stuff is the room centering part, where we could not find a better way of writing it.
- The data flow of the App is also quite complex. We use a `ViewModel` for maintaining core data structures. It may seem to be overly complex, but it helps manage the lifecycle. See `model/GameViewModel.kt` and the corresponding observers for detail.
- The room detail view adds more complexity to it. In order to switch between the detail view and the map, the player status must be handled carefully.

Item

- Most items are quite boring, but there are several ones that take longer to implement.
- For `activated` items like `Chest Fanatic`, after activating them, player needs to select a room. When the player holds multiple `activated` items, we need to know which one he/she is interacting with. See functions `onRoomClick`, `doActivate`, `onActivate`, `doRoomSelected` for how we implement this.
- `MapReduce` is the only `LEGENDARY` item in the game. It merges all enemies of a room into one by using `map` and `reduce`.
- An item has a rare level. They are all in the same pool, but with different probability to be chosen. The implementation is quite elegant, see functions `fetchItems` and `fetchItem`.

Battle Simulator

- Nothing special, a lot of `ifs` to check for end conditions. See `BattleSimulator.kt`.

Award System

- When a player opens a chest or wins a battle, an award is sampled and then applied.
- The code can be further cleaned using `reflection`, but we don't think it is necessary. See `Award.kt` and `AwardSampler.kt`.

Stage

- The reachability problem is a classic BFS (breadth-first search) problem. See function `canReach`.
- A small feature is that when the player has exhausted the item pool, function `tryQuickAccess` handles it elegantly.

General

- The player and stage changes all the time. Every time they change, the App will redraw everything to make sure that the player sees the latest view. It saves us tons of time. Although it may not scale well, for a small game like this it works perfectly. Never make optimizations too early.
- Since we may redraw the screen multiple times, it is crucial that these redraw functions MUST be ideopotent.
- It is the first time of writing a medium-sized application by scratch. We definitely learned a lot of advanced Kotlin features. Although it is a complete game, there are so many small features that is missing right now. Due to the limitation of time, we only have time to finish the core features.

Lines of Code

The total lines of Kotlin code is 1922, and the total lines of XML is 581.

Debug Story:

In our project, we use `addView()` to add 25 buttons in a FrameLayout called mapContainer. In order to make the game interface more tidy, I try to make these buttons appear in the center of the page. To achieve button centering, I need to get the width and height of the mapContainer.

At first, I used `mapContainer.height` to get the height of the component, but I failed. I found that the height is 0. `getWidth()` and `getHeight()` methods returns 0 when layouts width and height are set as match_parent and wrap_content. Dimensions are not measured.

I learned such a method on StackOverflow,

```
mapContainer.viewTreeObserver.addOnGlobalLayoutListener(
object :
OnGlobalLayoutListener {
override fun onGlobalLayout() {
mapContainer.viewTreeObserver.removeOnGlobalLayoutListener(this)
containerHeight = mapContainer.height
containerWidth = mapContainer.width
```

In this way, the width and height of the mapContainer can be successfully obtained.

After the painful debugging, I also learned another thing that in FrameLayout, the coordinate (0, 0) refers to the upper left corner of the FrameLayout, instead of the upper left corner of the entire screen. Besides `Log.d()` is our good friend.