



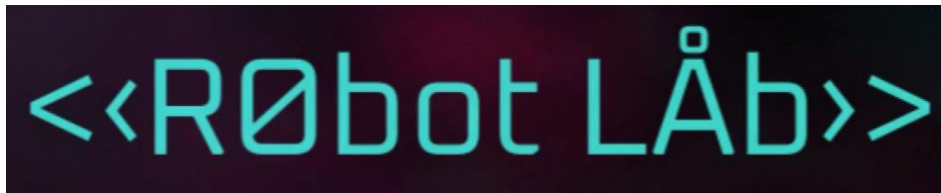
THE UNIVERSITY OF HONG KONG
FACULTY OF ENGINEERING

D E P A R T M E N T O F
COMPUTER SCIENCE

THE UNIVERSITY OF HONG KONG

DEPARTMENT OF COMPUTER SCIENCE

COMP3329 Computer Game Design and Programming



Game Report

Group 26

Contents

1. Overview	P.3
2. Scope of project	P.3-4
3. Game elements	P.4-5
4. Evaluation: The Four Element Evaluation	P.6-8
5. Technical aspect & Development process	P.9
6. Future plan	P.10
7. Conclusion	P.10
Appendix	P.11

**Warning: You need to install Unity's Cinemachine package before
running the game in Unity editor.**

1. Overview

<RØbot LÅb> is a single player 2D platformer/ coding game that takes inspiration from a website called [CodinGame](https://www.codingame.com/start) (<https://www.codingame.com/start>). This game combines elements of traditional platformer games and programming. In <RØbot LÅb>, player would select one out of 4 robots, each have different features, and instruct the robot via codes to reach the destination. This game aims at providing an enjoyable learning platform for beginner programmers and introducing the programmer's world to people who are interested in.

2. Scope of project

Feature scope

Initially the game was designed to be a traditional platformer game with a laser chasing the player and some enemies to stop player, and without the coding part. During the stage of game planning, I coincidentally found [CodinGame](https://www.codingame.com/start) and I thought that it might be interesting if I could do something similar and incorporate coding and platform game together. This is how this game idea born. The laser part was also abandoned due to time constrain, instead a timer is added in the gameplay to ensure player's robot could finish the game within certain time.

The game currently has 5 stages, where the first 3 stages are introductory stages/ tutorials. Players are given some functions like walk, jump and fire, and variables like the goal position, enemies' positions, and platform positions. In the first 3 stages, players could familiarize themselves with the coding environment and how these functions and variables works. The latter 2 stages would be a little bit more difficult for beginner programmers, but in general these stages are expected to be completed by players with enough trials and improvements.

Graphics & Audio

Due to time limitation, it is difficult to draw the sprites or make the audio which would be used in the game, therefore the Unity Asset Store was extensively used in the game's graphics. For a complete list of assets adapted from Unity Asset Store, see Appendix.

External resources

Various YouTube tutorial videos were used in developing the game, and MoonSharp was used as the interpreter to interpret players' code and link it to the robot. For a complete reference list, see Appendix.

3. Game elements

Some key screens are displayed in this section.

Robots



Fig.1 Robots information & descriptions in the game

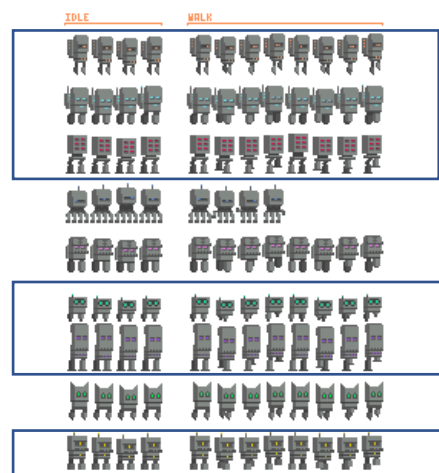


Fig.2 Sprite sheet for robots (bounded are used in game)

Screens

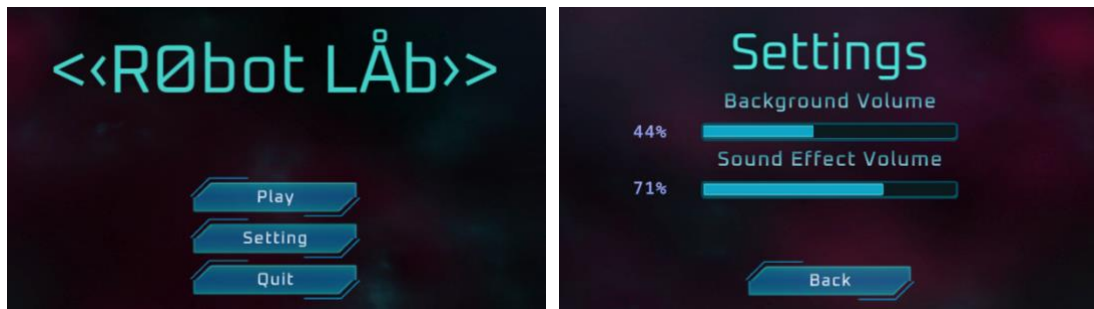


Fig.3&4 Main menu screen & Setting screen

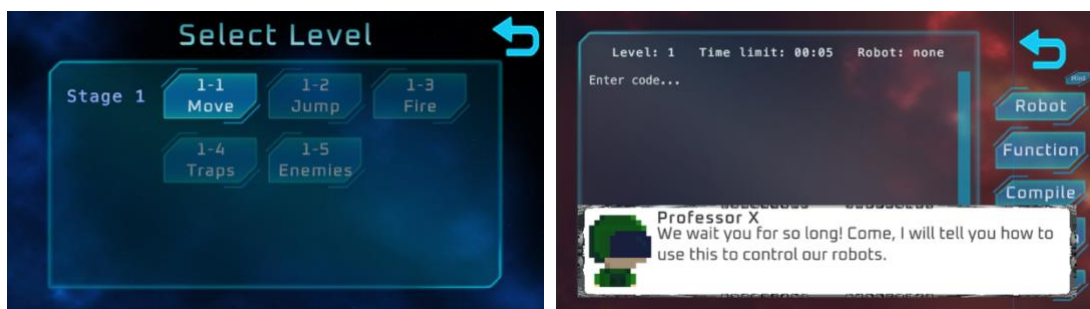


Fig.5&6 Select level screen & Coding screen (with dialogue)



Fig.7 Gaming screen (with console enabled)

4. Evaluation: The Four Element Evaluation

Technology

<RØbot LÅb> was designed for desktop users. It was developed using Unity Engine, and is capable to run on different platforms, such as MacOS and Windows. Players are expected to use keyboards to type in codes and control their robots in manual mode, and thus mobile phones are not preferred in playing the game.

Mechanics

The game integrates both programming and 2D platforming. It currently contains 5 levels. The ultimate goal of a level is to reach the destined goal position, in the *Game Scene* it is displayed as a door. During this process, there will be some obstacles to stop the player from reaching the goal, such as enemies and traps, and some utility objects to help players, such as floating platforms. Players would need to instruct the robot via codes and overcome all the hurdles to reach the goal within the time limit. Players can lose in three ways: Getting hit by obstacles (enemies and traps); Getting hit by the enemies' bullet; Cannot reach the target position within the time limit.

When player enters the *Coding Scene*, they are asked to: select a robot for running the level, write codes to instruct the robot, and run the codes to see the result. Players are suggested to investigate the level via *test running*, i.e. *going manual mode* each time they enter a new level. Afterwards, players are suggested to read the provided functions by clicking the Function button. Many default functions available in Lua are disabled and unexposed to player due to security issue. A list of functions provided, and its respective keys used in test running are showed in the following:

Key(s)	Function	Description
	walk(0)	Stay still
A / ←	walk(<0)	Walk towards the left
D / →	walk(>0)	Walk towards the right
J	fire()	Fire a bullet towards the current facing direction
K	jump()	Jump

Fig.8 Descriptions of keys used in test running and player functions

The game provides some variables for the player to use. They are useful in many situations, as they allow players to compare and make `if` statements genuinely meaningful to use. A list of variables provided is presented in the following:

Name	Type	Description
currentPos	Vector2	The current position of the robot
winningPos	Vector2	The goal position
enemiesPos	Vector2[]	The positions of enemies
platformPos	HorizontalBlock[]	The positions of platforms
trapPos	HorizontalBlock[]	The positions of traps
maxJumpDistance	Vector2	The maximum jumping distance and height of the robot on a flat surface

Fig.9 List of variables and their descriptions

The following list out the type of variables used in the game, and provides a brief description to each:

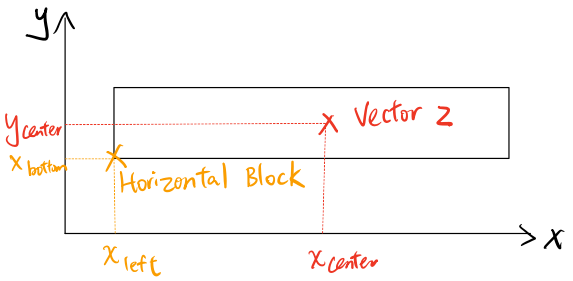
Type	Description
Vector2	Same to Unity's Vector2 type. A <i>copy</i> of the actual Vector2 object is passed to the player.
HorizontalBlock	<p>A self-defined struct which represents the position of a horizontal block. It contains 3 floats: xStart, xEnd, and y, indicating the start and end coordinates of the block. Unlike Vector2, the coordinates are located at the bottom left of the object. Illustrated as following:</p> 
[] (table)	Similar to Unity's array structure. It is automatically converted into Lua's table structure by the Moonsharp interpreter.

Fig.10 List of data types and their descriptions

The player can opt to enable console by clicking the respective button. The console acts as an area for functions like `print()` to print the message. For example, the player could type `print(currentPos.x)` to print the x-coordinate of the current robot position on the console. The console is updated each frame, so player could observe the information in real time, and is a helpful debugger for players to reflect on

their codes. Notice that if the console is off, functions like `print()` would be ignored.

Dialogues will appear in some levels when the players first enter the *Coding Scene* of that level. Two characters, **Professor X** and **John** will be guiding the players and explaining new game mechanics to them. The first level would have significantly more dialogues as the user interface and the coding environment would be explained to the players.

The first 3 levels are tutorials, therefore they are relatively simpler, and sufficient number of hints would be provided to players for completing the levels. The difficulty of level increases as player progress and reach higher level. If players are stuck in a level, they can click the hint button for some hints. The hints are presented in dialogues.

Story

<RØbot LÅb>'s story starts with the player, who act as a new student of **Professor X**, is assigned by him to test out his robots in the LÅb. Since **Professor X** is very busy, he asked his assistant **John**, to be the mentor of the player. Player's job is to find the best way to code **Professor X**'s robots and pass all the levels designed by him.

Aesthetics

The tone of the game is sci-fi futuristic to suit the storyline. To build such a style, the game mainly uses light blue mixed with metallic white/light grey as the color choice. The timer and console text in the *Game Scene* uses an extremely bright green, which is often referred as the "hacker color".

Different space/universe backgrounds are used in different scenes. Except for the *Game Scene*, all other scenes' background is dynamic, i.e. the background is constantly moving, so as to prevent player getting bored with static background. The background music is selected with the hope to bring tension to player without distracting them from the game. When players enter the *Game Scene*, the music is changed to a more intense style to bring out

Sound effects and animations of the buttons and robots make the game livelier and less dull.

5. Technical aspect & Development process

<<RØbot LÅb>> contains 4 scenes, which are *Menu Scene*, *Level Select Scene*, *Coding Scene*, and *Game Scene*. In general, *Coding Scene* and its interaction between *Game Scene* are the core and the most complex components of the game.

The development process started with building a usual 2D platformer game in *Game Scene*. Different layers of TileMap are used to separate different kinds of objects. For example, traps have their own layer of TileMap, otherwise it is not possible to separate them from other objects like walls and floating platforms. Object pooling is used to store the bullets of players and enemies. Singleton pattern is used in objects that only one instance is allowed.

Before building the *Coding Scene*, a considerable amount of time was spent to search for a suitable in-game compiler/ interpreter, and eventually MoonSharp was selected. It allows runtime compiling from string to code via `Script.DoString()`, and is therefore the core of this game. The functions and variables are passed to the player's script via MoonSharp's functions. The class `RobotInfoForGame` acts as a bridge for information flow between the *Coding Scene* and the *Game Scene*. The codes and other related information would be saved by the *Coding Scene* UI into `RobotInfoForGame`, and the game manager in *Game Scene* would process the information passed.

The *Menu Scene*, *Level Select Scene*, and *Coding Scene* are built with reverse order, and all of them are full of UI elements. Typical UI elements such as Image, Text, Button, Slider, Scrollbar, and Input Field are used. During this process, more levels are added to the game and some general utility functions are added, such as the scene transition effects and the save/load module.

The audio and the dialogue system were the last to be added since they do not affect the main game logic.

6. Future plan

Due to time limitation, many features are unfortunately cut and there is plenty of room for improvement. A list of few possible future improvements are as follows:

1. Implement Cheato. Cheato is a robot that provides extra functions to player for easier level completion. However, the implementation was not completed due to time constraints.
2. Add more diversity to levels. Current levels only consist of ground, wall, floating platforms, goal door, enemies, and traps. New objects such as humans (if robot shoots human, then level lose) and helpful items (like when pick up, invincible for a few seconds) can be added.
3. Provide more available functions and variables to players.
4. Improve the story of the game and the dialogue system to make the game more interactive.
5. When players are test running in manual mode, allow them to be invincible, so that they do not die and respawn, because some levels might be too extreme for human to complete.
6. Implement a score system instead of a hard time limit which makes players lose immediately if they cannot reach it.
7. Convert `HorizontalBlock` to `Block` which have four variables `x1`, `x2`, `y1`, `y2` each locate the start and end of x & y, instead of just `xStart`, `xEnd`, and `y`.
8. Camera full map view by pressing button.

7. Conclusion

<<RØbot LÅb>> is a short project done in about 3 months' time. It was a watered-down version compared to my initial idea, yet I was satisfied about the result. The main theme of the game – to make programming more enjoyable to learn – was well brought out. The game will be consistently updated and improved in the future. It was hoped that players could also enjoy the game while practicing their logical thinking skill.

Appendix

Unity Packages

Cinemachine

Unity Asset Store

MoonSharp interpreter:

<https://assetstore.unity.com/packages/tools/moonsharp-33776>

Sprites:

<https://assetstore.unity.com/packages/2d/characters/free-pixel-army-platformer-pack-168264>

<https://assetstore.unity.com/packages/2d/textures-materials/space-star-field-backgrounds-109689>

<https://assetstore.unity.com/packages/2d/textures-materials/dynamic-space-background-lite-104606>

UI:

<https://assetstore.unity.com/packages/tools/gui/alpha-flashing-ui-ui-shaders-164360>

<https://assetstore.unity.com/packages/2d/gui/sci-fi-gui-skin-15606>

<https://assetstore.unity.com/packages/2d/gui/icons/sci-fi-interface-frames-10747>

Audio:

<https://assetstore.unity.com/packages/audio/sound-fx/ui-sfx-36989>

<https://assetstore.unity.com/packages/audio/music/dynamic-music-35925>

<https://assetstore.unity.com/packages/audio/sound-fx/sci-fi-sfx-32830>

External references

Menu:

https://www.youtube.com/watch?v=zc8ac_qUXQY

Platformer:

<https://www.youtube.com/watch?v=TcranVQUQ5U&list=PLgOEwFbvGm5o8hayFB6skAfa8Z-mw4dPV>

<https://www.youtube.com/watch?v=jj1km9Nd4bU&list=PLX2vGYjWbI0REfhDHPpdIBjjrzDHDP-xT&index=4>

Dialogue system:

https://www.youtube.com/watch?v=_nRzoTzeyxU

UI:

https://www.youtube.com/watch?v=_RIsfVOqTaE

Scene transitions:

<https://www.youtube.com/watch?v=CE9VOZivb3I>

Level Selector:

<https://www.youtube.com/watch?v=-cTgL9jhpUQ>

Audio:

<https://www.youtube.com/watch?v=6OT43pvUyfY>

Game trailer audio:

https://www.youtube.com/watch?v=60llyQkMces&list=PL7-_Ltr8Xtuy90yqoxcvm_D_rZGHYQJ-3&index=10

Save/Load:

<https://www.red-gate.com/simple-talk/development/dotnet-development/saving-game-data-with-unity/>

Miscellaneous:

<https://forum.unity.com/threads/static-variables-and-editor-reload-behavior.615154/>

<https://gist.github.com/ProGM/9cb9ae1f7c8c2a4bd3873e4df14a6687>

<https://www.youtube.com/watch?v=oNz4I0RfsEg>

Solutions

----- Lv1 -----
(Robo)

walk(1)

----- Lv2 -----
(Jumpo)

walk(1)
jump()

----- Lv3 -----
(Robo)

walk(1)
fire()

or

walk(1)
jump()

----- Lv4 -----
(Robo)

```
for i = 1, #trapPos do
    local tPos = trapPos[i]
    local distx = currentPos.x + maxJumpDistance.x
    if (currentPos.x <= tPos.xStart and distx >=
tPos.xEnd + 1.7) then
        jump()
    end
end
walk(1)
```

```

----- Lv5 -----
(Robo)

if (#enemiesPos != 0) then
    walk(-1) fire()
else
    isTrap = false
    for i = 1, #trapPos do
        local tPos = trapPos[i]
        local distx = currentPos.x + maxJumpDistance.x
        if (currentPos.x <= tPos.xStart or distx <=
tPos.xEnd) then
            isTrap = true
        end
    end
    if not isTrap then
        jump()
    end
    walk(1)
end

```