Department of Computer Engineering

BLG 439E

Computer Project I

Mobile Game Development

A Game to Teach Children How to Code

150130109 - Güllü Katık

150140141 - Güneş Yurdakul

150130039 - Ozan Ata

04.12.2017

1. Introduction and Project Description

In this project we designed and implemented a puzzle game to teach programming logic to kids between ages 6 to 11. The aim of the game is to make the unicorn collect candies on a 6*6 board. Our game allows children to learn basic programing concepts such as sequencing, functions and loop.

Unlike other games, the movements and commands are not synchronous, the player lines up all the commands before the movement begins, so the child learns sequencing the most important part of coding. Repetitive movements will help the children to gain the ability of using functions and loops.

2. Background

Before, we designed the puzzles we found two games for kids which are developed to teach how to code and analyzed them. These are “Coding Games For Kids – Learn To Code With Play” and “Lightbot: Code Hour”. Coding Games For Kids have more levels than Lightbot and it is more successful in giving loop logic. In Lightbot recursive function is used instead of loops, this may cause children to mislearn loop logic. For all this reasons we took Coding Games For Kids as a model while designing our game.

We used Ionic platform for implementation of the game. Ionic is an open source software development kit that enables hybrid mobile application development. Developers can build application using web technologies like AngularJS, HTML5, CSS, and Sass; and then the applications can be installed to mobile devices by means of Cordova, can also be run on browsers.

3. How to Play

The game screen consists a 6*6 board that contains a unicorn and candies. Under the board there are commands such as directions, collect candy button, loop and function button. When user clicks a command, the command is added to the commands list(first line under the commands which has the start button on its left. If the user clicks on the second line the commands will be added as function commands and if user clicks back on the first line and chooses blue function button. The function will be added to the main commands list. The third line is for defining loops. The number of iterations can be increased by clicking the red button on the third line. The basket button is for collecting the candies.

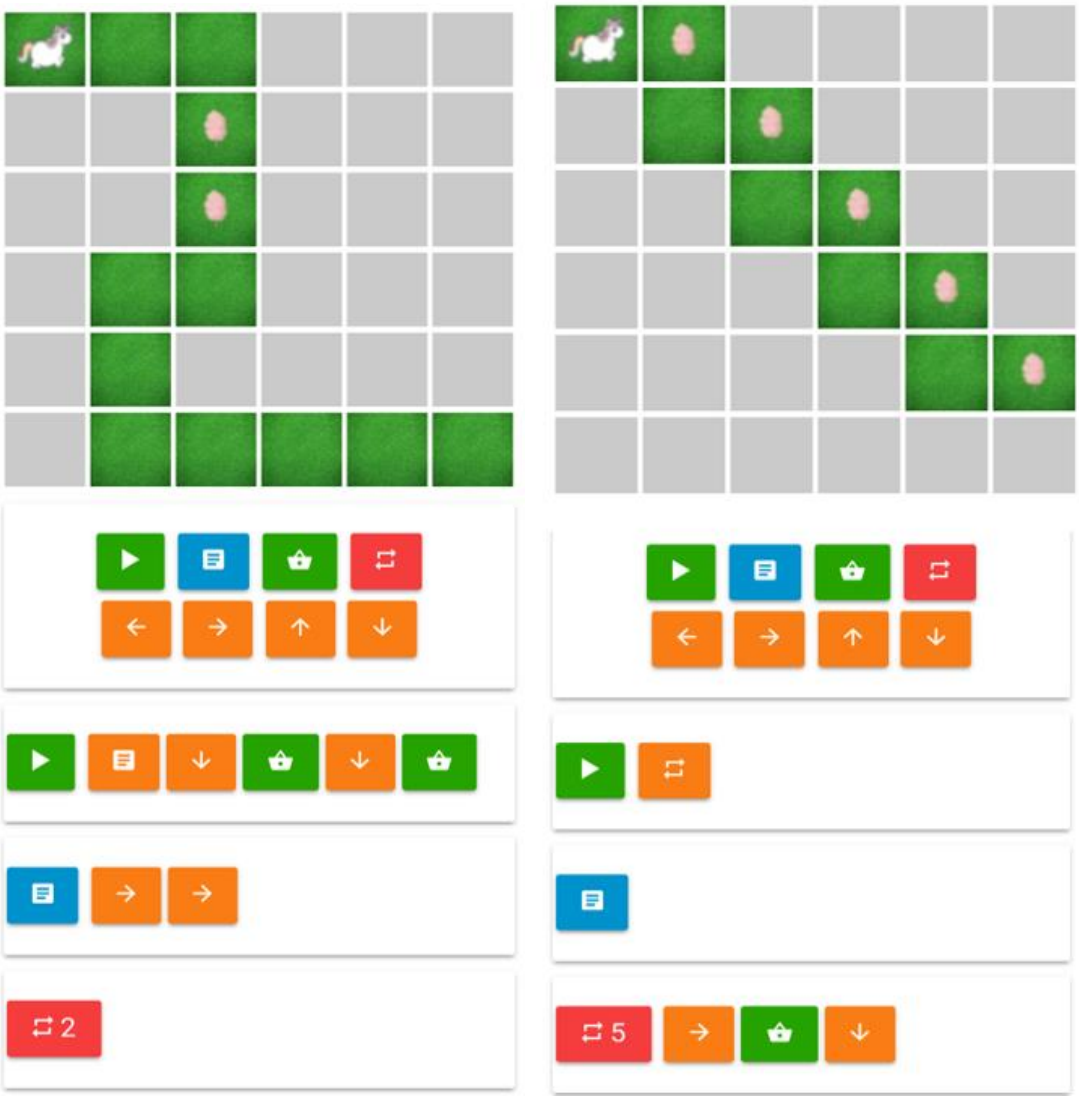
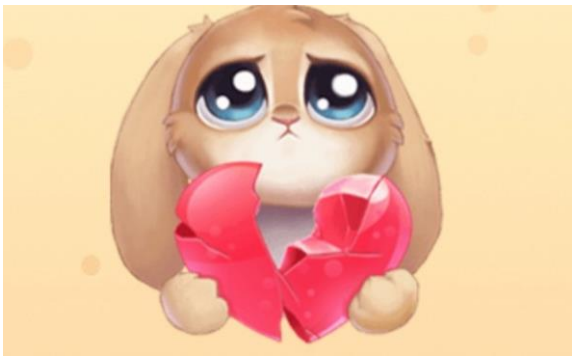


Figure 1: Level 1 on left, level 3 on right

Player must move the unicorn to cells that contain candy using left, right, up, and down commands. Loop or function must be used for repeated movements like on the Level 3. Loop count can be increased by clicking on it when necessary. When the player completes the level, a rainbow image is opened on the screen and the next level gets started. Otherwise the board is reset and the level is replayed.



4. Implementation

Levels hold in a three dimensional array. Each cell has a color, the green path as black, blank cells as grey and the cells that contain candy as red holds in the array.

```
constructor(public navCtrl: NavController, public navParams: NavParams) {      You, 27 minutes ago • Uncommitted change
  if (navParams.get('item'))
    this.level = navParams.get('item');
  else
    this.level = 0
  this.x = 0
  this.y = 0
  this.nMove = 0
  this.levels = [
    [[new Block("black_active"), new Block("black"), new Block("black"), new Block("grey"), new Block("grey"), new Block("grey")]
      , [new Block("grey"), new Block("grey"), new Block("red"), new Block("grey"), new Block("grey"), new Block("grey")]
      , [new Block("grey"), new Block("grey"), new Block("red"), new Block("grey"), new Block("grey"), new Block("grey")]
      , [new Block("grey"), new Block("black"), new Block("black"), new Block("grey"), new Block("grey"), new Block("grey")]
      , [new Block("grey"), new Block("black"), new Block("grey"), new Block("grey"), new Block("grey"), new Block("grey")]
      , [new Block("grey"), new Block("black"), new Block("black"), new Block("black"), new Block("black"), new Block("black")]
    ],
    [[new Block("black_active"), new Block("black"), new Block("black"), new Block("black"), new Block("black"), new Block("red")]
      , [new Block("grey"), new Block("grey"), new Block("grey"), new Block("grey"), new Block("grey"), new Block("black")]
      , [new Block("red"), new Block("black"), new Block("black"), new Block("black"), new Block("black"), new Block("red")]
      , [new Block("black"), new Block("grey"), new Block("grey"), new Block("grey"), new Block("grey"), new Block("grey")]
      , [new Block("black"), new Block("grey"), new Block("grey"), new Block("grey"), new Block("grey"), new Block("grey")]
      , [new Block("red"), new Block("black"), new Block("black"), new Block("black"), new Block("black"), new Block("red")]
    ],
    [[new Block("black_active"), new Block("red"), new Block("grey"), new Block("grey"), new Block("grey"), new Block("grey")]
      , [new Block("grey"), new Block("black"), new Block("red"), new Block("grey"), new Block("grey"), new Block("grey")]
      , [new Block("grey"), new Block("grey"), new Block("black"), new Block("red"), new Block("grey"), new Block("grey")]
      , [new Block("grey"), new Block("grey"), new Block("grey"), new Block("black"), new Block("red"), new Block("grey")]
      , [new Block("grey"), new Block("grey"), new Block("grey"), new Block("grey"), new Block("black"), new Block("red")]
      , [new Block("grey"), new Block("grey"), new Block("grey"), new Block("grey"), new Block("grey"), new Block("grey")]
    ],
    [[new Block("black_active"), new Block("black"), new Block("red"), new Block("grey"), new Block("grey"), new Block("grey")]
      , [new Block("grey"), new Block("grey"), new Block("black"), new Block("grey"), new Block("grey"), new Block("grey")]
      , [new Block("red"), new Block("black"), new Block("red"), new Block("grey"), new Block("grey"), new Block("grey")]
      , [new Block("black"), new Block("grey"), new Block("grey"), new Block("grey"), new Block("grey"), new Block("grey")]
      , [new Block("red"), new Block("black"), new Block("red"), new Block("grey"), new Block("grey"), new Block("grey")]
      , [new Block("grey"), new Block("grey"), new Block("black"), new Block("grey"), new Block("grey"), new Block("grey")]
    ]
  ]

  this.blocks = this.levels[this.level]
}
```

Player must be click to loop/function section to add command to loop/function.

```
<ion-card padding-vertical (click)="set_function()">
  <ion-list>
    <button ion-button color="blueCustom">
      <ion-icon name="list-box"></ion-icon>
    </button>
    <button ion-button *ngFor="let item of function_command_names" color={{get_color(item)}} (click)="delete_function(item)">
      <ion-icon name="{{item}}"></ion-icon>
    </button>
  </ion-list>
</ion-card>

<ion-card padding-vertical (click)="set_loop()">
  <ion-list>
    <button ion-button item-left color="danger" (click)="set_loop_count()">
      <ion-icon name="repeat">
        {{loop_count}}
      </ion-icon>
    </button>
    <button ion-button *ngFor="let item of loop_command_names" color={{get_color(item)}} (click)="delete_loop(item)">
      <ion-icon name="{{item}}"></ion-icon>
    </button>
  </ion-list>
</ion-card>
```

When player select a command (for example left), first it is checked whether the command is added to function, then it is checked whether the command is added to loop, if the command is not added to neither of them, it is added to main function as default.

```
left() {
    if (this.is_function) {
      this.function_list.push([0, -1]);
      this.function_command_names.push("arrow-back");
    }
    else if (this.is_loop) {
      this.loop_list.push([0, -1]);
      this.loop_command_names.push("arrow-back");
    }
    else {
      this.list.push([0, -1]);
      this.command_names.push("arrow-back");
    }
  }
```

While commands of main function are held command_names list, there are function_command_names list for function commands and loop_command_names list for loop commands. Function and loop commands are held these lists until execution.

When the player press play button the execution method is called. This method executes all commands in the command list one by one until no command left in the list. It checks whether the level is completed after each execution.

```
execute() {
  var interval = setInterval(() => {
    if (this.list.length !== 0) {
      if (this.list[0] === "function" || this.list[0] === "loop") {
        this.reshape()
      }
      let command = this.list[0]
      let result = this.move(command[0], command[1])

      this.list.splice(0, 1)
      this.command_names.splice(0, 1)
      if (result || this.nMove > 30) {
        if (this.level < 3 && result)
          this.level = this.level + 1

        clearInterval(interval)
        this.reload()
        this.navCtrl.push(ResultPage, {
          item: [this.level, true]
        });
        return
      }
    }
    else {
      clearInterval(interval)
      this.reload()
      this.navCtrl.push(ResultPage, {
        item: [this.level, false]
      });
      return
    }
  }, 500);
}
```

When the execution sequence reaches a function or a loop, reshape method is called. The method adds function/loop commands to main command list.

```
reshape() {
  if (this.list[0] === "function") {
    this.list.splice(0, 1)
    for (let i = this.function_list.length - 1; i >= 0; i--) {
      this.list.unshift(this.function_list[i])
    }
  }
  else {
    this.list.splice(0, 1)
    for (let j = 0; j < this.loop_count; j++) {
      for (let i = this.loop_list.length - 1; i >= 0; i--) {
        this.list.unshift(this.loop_list[i])
      }
    }
  }
```