École Polytechnique Fédérale de Lausanne
EPFL-CS107 - Introduction à la programmation

Conception - MINIPROJET-2

Students:
BRUNO Elie
KUGLER Sebastian

Teacher:
Jamila SAM

Thursday 15th December, 2022

Contents

Conception

1.1 Étape 1 :

1.1.1 ICRogueRoom :

The ICRogueRoom class is a subclass of the Area class from the areagame package. It represents a single room in a game called "ICRogue". The room has four Connector objects, which can be opened and closed, and the room has a behaviorName that specifies the behavior of the room. The ICRogueRoom class also implements the Logic interface, which defines a isOn() method that returns true if the room has been visited.

1.1.2 Projectile

The Projectile class represents a projectile that can be fired by the player or enemies in the game ICRogue. It has attributes such as move duration, damage, and a consumed flag to track its state. It also implements the Consumable and Interactor interfaces, allowing it to be consumed and interact with other objects in the game. The Projectile class has several constructors to initialize the projectile with different parameters such as damage and move duration. It also has an update method to update its state and a draw method to render it on the screen. However, the Projectile class is an abstract class, meaning that it cannot be instantiated directly and must be extended by a subclass that provides a specific implementation of the projectile. For example, the Fire class extends the Projectile class and provides a fireball implementation.

N.B : This class has two subclasses that are Fire.java and Arrow.java (we will come back later to Arrow.java)

Fire

The Fire class represents a fireball that can be used as a weapon in the game ICRogue (when X is pressed). It has a duration and a damage attribute that determine how long it stays on the screen and how much damage it does to an enemy when it hits them. The Fire class also has an animation and a sound effect that are played when the fireball is created and moves around on the screen. The Fire class extends the Projectile class, which provides it with basic movement and collision detection capabilities. It also implements the Acoustics interface, allowing it to produce sound effects when it is invocked.

1.1.3 Objets à collecter & Interactions :

Item :

This is a base class for items in a game. It extends CollectableAreaEntity, which means that it is an entity that can be picked up by the player in the game. The class has a Boolean field called isPickedUp that indicates whether the item has been picked up or not. It also has a Sprite field called sprite that represents the visual appearance of the item. The draw method is implemented

to draw the item's sprite if it has not been picked up. The acceptInteraction method, which is used to handle interactions with the item, is left abstract because the specific behavior of the item will depend on the type of item it is.

Cherry This class has been coded to fit the project description but it is not implemented

Heart (Extension) The Heart class represents a heart item in a game. The item has a sprite with an animation that is updated and drawn on the canvas. The class also overrides the acceptInteraction method to accept interactions from an ICRogueInteractionHandler. This method is called when the player interacts with the heart item, such as picking it up.
Note that it will always be picked up but it will only heal the player until he reaches his maximum healthpoints. It gives the player 5 hp and has a 45 % chance of spawning inside a turret room at rondom coordinates

Key The Key class represents a key item in a game. The key has a sprite and a unique ID. The class overrides the acceptInteraction method to accept interactions from an ICRogueInteractionHandler. This method is called when the player interacts with the key item in some way, such as picking it up or using it to open a door. The getId method can be used to get the unique ID of the key so we can check this ID with connectors.

Staff The Staff class represents a staff item in a game. The staff has a sprite with an animation that is updated and drawn on the canvas. The class overrides the acceptInteraction method to accept interactions from an ICRogueInteractionHandler. This method is called when the player interacts with the staff item in some way, such as picking it up or using it to start throwing fireballs. The isViewInteractable and takeCellSpace methods indicate that the staff can be interacted with and occupies a cell on the game grid, respectively.

1.2 Étape 2 :

1.2.1 Connectors :

The Connector class represents a connector in a game, which is an entity that allows the player to switch between rooms in an area. The connector has a sprite and a state that determines whether it is open, closed, locked, or invisible. The connector can be opened with a key if it is locked, or simply opened if it is closed. The connector can also be set to have a destination area and coordinates, which the player will be transported to when the connector is used. The class also overrides the acceptInteraction method to accept interactions from an ICRogueInteractionHandler, which is likely called when the player interacts with the connector.

1.3 Étape 3 :

1.3.1 Enemy Room

The Level0EnemyRoom class represents an enemy room in a game. This class extends the Level0Room class, which is a room in an area. The Level0EnemyRoom class adds the ability to contain enemies, which are added to a list. The update method overrides the update method

in the superclass to update the enemies in the room as well as the room itself. The createArea method overrides the createArea method in the superclass to add the enemies to the area when it is created. The isOn method overrides the isOn method in the superclass to return true only if all of the enemies in the room are dead. This is likely used to check if the player has defeated all of the enemies in the room before allowing them to proceed.

Turret Room :

The Level0TurretRoom class represents a room in a game that contains two turrets and potentially a heart item. This class extends the Level0EnemyRoom class, which represents a room that contains enemies. The Level0TurretRoom class adds two turrets to its list of enemies, and it also creates a heart object that is potentially added to the game. The begin method overrides the begin method in the superclass to register the heart object with the game if a random number is less than 45. The Level0TurretRoom constructor initializes the two turrets and the heart, and adds them to the list of enemies.

1.3.2 Enemy

The Enemy class represents an enemy in a game. It extends the ICRogueActor class, which means that it can exist and be drawn on an Area in the game. The Enemy class has a boolean flag isAlive that keeps track of whether the enemy is alive or not, and a Sprite object that can be used to draw the enemy on the screen. The class also has a Consume method that sets the isAlive flag to false and unregisters the enemy from the Area it belongs to.

Turret :

The Turret class extends the Enemy class and represents a turret enemy in the game. The Turret class has a ICRogueTurretInteractionHandler object that handles interactions with the Turret. The Turret class also has an array of Orientation objects that specify the directions in which the Turret should fire arrows. The Turret class overrides the update method to fire arrows in the specified directions at regular intervals. The Turret class also implements the Interactor interface, which means that it can interact with other Interactable objects in the game. The Turret class has a wantsCellInteraction and wantsViewInteraction method that both return true, indicating that the Turret should respond to cell and view interactions. The Turret class also has an interactWith method that allows other Interactable objects to interact with the Turret's interaction handler.

1.3.3 Bonus :

End Message :

We added an ASCII Art at the end of the game, If the game is lost then, Game Over is printed in the console and then it finished by showing credentials which is our names.7

1.4 Étape 4 : Pseudorandom Room Generation

The suggested division into the overall geographic-, followed by the specific room placement and -creation was realized in our code structure.

1.4.1 Stage 1: Map Rooms

A quadratic map of sufficient size is generated, initialized (`mapRooms`) and populated pseudo-randomly with adjacent room placeholders (`fillMap`). The map is a two-dimensional array of `MapState`, an enum-type abstraction of - and basis for - the real `ICRogueRoom` array filled in the second stage. Helper functions of this stage are self-descriptively named `fillMapAround`, `randomPlacedPosition` and `freeNeighbourSlots`.

1.4.2 Stage 2: Create specific Rooms

Based on the map of the previous stage, `placeRooms` places the specific rooms defined by the respective `LevelX` class at pseudorandom, free coordinates. `RandomHelper.chooseKInList()` was thus modified to be able to choose from a list of `DiscreteCoordinates`. The creation of each room is delegated to the specific `LevelX` class and the boss room is created last to prevent it being an obligatory transit room on the map.

1.4.3 Modularity

The level-generating code is crafted in such a way that an arbitrary number of levels can be easily generated. Although not part of this project, other `LevelX` classes could be trivially implemented and used in harmony with the Level.java class and its generation algorithms.

1.5 Étape 5 : Extensions

1.5.1 Animations :

We decided to animate all of our players, the "pet" following us, the main player, the heart spawning inside the turret room, the staff which has a sort of fire animations next to it and lastly the fireball.
The following method needed to be coded to animate the following character : "boy.png"

Extract col method

This method has been coded in the Sprite.java file to allow us to animate a pet that follows us.
We coded this because, the given extractSprites method only took images where all the positions where line by line but lot of images that could be found in the res folder had their positions column by column so the first method was not working.

```
1  public static Sprite[][] extractSpritesCol(String name, int nbFrames, float width
     , float height,
2                          Positionable parent, int regionWidth, int regionHeight,
     Vector anchor, Orientation[] order) {
```

```
3
4     Sprite[][] sprites = new Sprite[4][nbFrames];
5
6     for (int i = 0; i < nbFrames; i++) {
7       int j = 0;
8       sprites[order[0].ordinal()][i] = new Sprite(name, width, height, parent,
9           new RegionOfInterest(regionWidth * j++, regionHeight * i, regionWidth,
10              regionHeight), anchor);
11      sprites[order[1].ordinal()][i] = new Sprite(name, width, height, parent,
12          new RegionOfInterest(regionWidth * j++, regionHeight * i, regionWidth,
13              regionHeight), anchor);
14      sprites[order[2].ordinal()][i] = new Sprite(name, width, height, parent,
15          new RegionOfInterest(regionWidth * j++, regionHeight * i, regionWidth,
16              regionHeight), anchor);
17      sprites[order[3].ordinal()][i] = new Sprite(name, width, height, parent,
18          new RegionOfInterest(regionWidth * j, regionHeight * i, regionWidth,
19              regionHeight), anchor);
20    }
21    return sprites;
22  }
```

1.5.2 Following character

We added a small character that is sticked to us, that is non playable and does not take any damage, it is also animated as previously stated.

1.5.3 HealthPoints :

We added the HealthPoints text above the player's head and we implemented an HPBar which is coded using the HPBar sprite located in the res folder, we added a red image that is being shrunk when the player loses HP.

1.5.4 Sound Implementation :

We added three sound implementations that are :

1. The arrow sound when arrows are being ejected of the turret.
2. The touchedPlayer sound, which activates when an arrow interacts with the player.
3. The fireBall sound, which is played when a fireball is created.

1.5.5 Design :

The player now walks on a shadow, it is similar to the shadow pokemon trainers used to have.

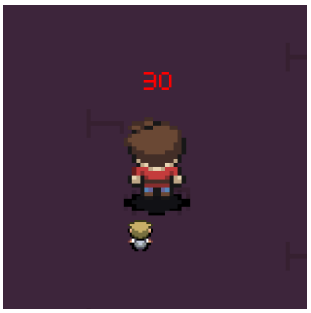

Figure 1.1: A nice shadow under the player