Object Oriented Programming
Programming report
final project:
2D Platformer

Michiel de Jong Jelle Visser
s2550768 s2238160

June 10, 2014

1 Problem description

The goal of this project was to make a Terraria-like 2D-platformer. We wanted the player to move around in a 2D-world, inhabited by monsters, and be able to change this world in game (like in Terraria). In short, our goal was to implement the following features:

- A 2D side-scrolling map, loaded from an image file.
- The user being able to create their own map and use it
- A player character with health and an inventory
- Enemies which the player can kill and that can kill the player, and drop an item upon death
- Doors that can be opened and closed
- Awarding a score to the player.

2 Problem analysis

A game consists of many different aspects and most of them can be seen as individual problems. Therefore, the divide and conquer strategy is a good method to solve this. The problems we had to tackle are the following:

- **Being able to render a map from an image file**
  The user can pass a new map as an argument, made in a program like Gimp or Photoshop. If he doesn't, a default map is provided. The properties an object gets should depend on its color in the original image file.
- **Implementing the user interface using a Model-View-Controller architecture**
  The model keeps track of what's going on in the game, which objects are in

place, what the current state is etc. The view should observe this model, rendering the current state on the screen. Finally, the controller manages user input such as walking around and performing actions.

- **Drawing the game world and its components**
  The whole map shouldn't be displayed at once. Instead, we want it to scroll with the player. Therefore, the panel that displays the game should keep track of the player's position and move such that he stays in the middle of the screen. We also want to implement a timer, which redraws the game world on a set frequency.

- **Making a player character that can interact with the world**
  The player has 100 health points and gets damage from touching monsters. When this amount reaches 0, we want the game to end.

- **Implementing enemies and creating an AI**
  The AI shouldn't be too advanced, but it should be able to at least walk around and damage the player, turning around when colliding with an object.

- **Handling collisions and physics**
  The player shouldn't be able to clip through ground and wall tiles, and the edges of the map. Each ground block in this game is a separate object, so calculating for each redrawing whether the player is touching any of these, or other, objects would be too taxing on resources. Instead, we want the game to only check this for nearby objects.

  We want to make a distinction between two types of in-game objects: objects that move and objects that are static. Moving objects can pass through each other, but not through static ones. They should therefore inherit from different classes.

  When in-air, a gravity function should provide a downward acceleration. The player should not be able to jump while in the air, i.e. is not touching a block below him.

- **Maintaining a score system**
  The final score is calculated upon reaching the goal, and is a function of the player's health and the time it took to get to the goal.

3 Program design

For the game we used the Observer pattern. We will explain the design by explaining each component of the MVC-architecture.

3.1 Game model

The *Game* class holds all data for the model. It loads all the placeable objects, and puts them in a $CollisionManager$. The $CollisionManager$ keeps track of which objects are in which area, so when an object has to check whether it collides with another object, it doesn't have to check all the objects in the map, but it can just check the objects in the neighbourhood. The *Placable*

objects all observe the *StepManager*, which notifies its observers 30 times each second. Each and every object in the game world is a *Placable*. Static objects, like simple blocks, inherit directly from this class. The class *Movable* extends *Placable*. Movable objects can move in the game world and experience physics, more specifically: gravity and friction. The *Player* and *Monster* classes inherit from this class. Also note that the *Player* class implements the interface *Actor*, meaning it can pick up items (currently only keys).

When the *StepManager* notifies its observers, it also passes the *InputManager*, which can tell if whether a key is down or up.

3.2 Game view

The class *GamePanel* is an observer of the Game class. Every time the model changes, the *GamePanel* repaints itself. The panel is attached to a game window. More specifically: it is attached to a card container. This is due to the card layout nature of our game. There are three different main panels: one for the game, one for the main menu and one for the how-to-play-section. Only one of these is displayed at a time. The card layout makes sure the right panel is displayed at the right time.

Sprite is a class which takes care of how an object is drawn. Its constructor can either take an image file or a set of integers. In the last case, it creates a simple colored rectangle from the parameters.

We have two special cases of classes inheriting from *Placable*: doors and stoppers. Stoppers are invisible blocks created around the map to prevent the player from leaving the game worlds. Doors can be opened and therefore have two different sprites: one for when its opened, and one for when it's closed.

3.3 Game controls

For handling input we make use of the class *InputController*. It contains two arrays, one to keep track of buttons that are currently down, and one to keep track which buttons have been pressed since the last time they were.

The menu buttons use simple action listeners to determine the correct action. The "Start game" button makes the card layout switch to the *GamePanel*, the "How to play" switches it to the *Descriptionpanel* and the "Quit game" button, of course, closes the game.

4 Evaluation of the program

In the end we made a completely functioning game. It is, however, different from our original plan of making a Terraria-esque game. This game behaves more like a classical platformer. While we deviated from our original plan, this game meets most of the minimum criteria of the assignment. The only things missing are the ability to save highscores and health-increasing items. We did not implement the ability to influence the map in-game nor did we implement loot and gear. Instead, the player now has to reach the goal position as fast as possible while dodging enemies and finding keys to open doors.

During the very final stages of development, a quite annoying bug came up: after pressing the "Start game" button, the game accepts no input. This is,

however, solved by opening another window and then opening the game window again. The origin of this bug is unknown, and could not be solved in time.

5 Conclusions

The reason we did not meet our planned goals and deviated from the original is due to underestimating the time needed for the project. Despite this, we encountered relatively little problems. During the process we learned a lot about programming games from scratch.

In the end, we delivered a working game with a source code that makes good use of OOP-principles.

6 Appendix: source code

6.1 Actor

```
1  /*
2   * An Actor object should be able to interact with
        other objects, and to pick up items.
3   * In this case, the Player object is the only actor
4   */
5  interface Actor {
6
7     public boolean take(Item item);
8
9     public boolean hasItem(String itemType);
10
11    public Item loseItem(String itemType);
12 }
```

6.2 Block

```
1  /*
2   * These are the solid blocks the world is made of
3   */
4
5  abstract class Block extends Placable {
6
7
8
9     public boolean isSolid(){
10       return true;
11    }
12 }
```

6.3 CollisionManager

```
1  import java.util.Set;
2  import java.util.HashSet;
3  /*
4   * A CollisionManager object keeps track of which
        objects are at a location.
5   *
6   * The set method stores the object in a set on every
        square that the object (partially) covers.
7   * The remove method removes the object from that area
        . It is important that the location and size of
8   * the object don't change between setting this object
         on the field and removing it.
9   *
10  * The get method returns a set of object that could
        in the specified area.
11  * This set may contain objects that are not exactly
        in the area, but if an object is in the area, it
        will be in the set
12  * This way, for exact collision checking not all the
        objects in the map have to be checked, but only
        the objects that are near
13  *
14  */
15 class CollisionManager {
16    private int width, height;
17
18    private Set<Placable>[][] field;
19
20    public CollisionManager(int w, int h){
21       width = w;
22       height = h;
23       field = new Set[width][height];
24       for (int i=0; i<width; i++){
25          for (int j=0; j<height; j++){
26             field[i][j] = new HashSet<Placable>();
27          }
28       }
29    }
30
31    public void set(Placable o){
32       for (int i = Math.max((int) o.getX(), 0); i <
            Math.min(Math.ceil(o.getX()+o.getWidth()),
            width); i++){
33          for (int j = Math.max((int) o.getY(), 0); j <
               Math.min(Math.ceil(o.getY()+o.getHeight()
            ),height); j++){
34             field[i][j].add(o);
35          }
```

```
36          }
37      }
38
39      public void remove(Placable o){
40          for (int i = Math.max((int) o.getX(), 0); i <
                Math.min(Math.ceil(o.getX()+o.getWidth()),
                width); i++){
41              for (int j = Math.max((int) o.getY(), 0); j <
                     Math.min(Math.ceil(o.getY()+o.getHeight()
                    ),height); j++){
42                  field[i][j].remove(o);
43              }
44          }
45      }
46
47      public Set<Placable> get(double x, double y, double
            w, double h){
48          Set<Placable> objects = new HashSet<>();
49          for (int i = Math.max((int) x, 0); i < Math.min(
                Math.ceil(x+w),width); i++){
50              for (int j = Math.max((int) y, 0); j < Math.
                    min(Math.ceil(y+h),height); j++){
51                  objects.addAll(field[i][j]);
52              }
53          }
54          return objects;
55      }
56
57
58  }
```

6.4 Controller

```
1   import java.util.Observable;
2   import java.awt.event.KeyListener;
3   import java.awt.event.KeyEvent;
4
5   /*
6    * The InputController is used to keep track which
         keyboard keys are pressed and down.
7    * The isDown method returns whether the key with the
         given key code is down at the moment,
8    * and the isPressed method returns whether the key
         with the given key code has been pressed
9    * since the last time that isPressed was called for
         that key.
10   *
11   */
12
```

```
13 class InputController extends Observable implements
       KeyListener{
14    private boolean[] down = new boolean[255];
15    private boolean[] pressed = new boolean[255];
16
17    public void keyPressed(KeyEvent e){
18       down[e.getKeyCode()]=true;
19       pressed[e.getKeyCode()]=true;
20    }
21    public void keyTyped(KeyEvent e){
22
23    }
24    public void keyReleased(KeyEvent e){
25       down[e.getKeyCode()]=false;
26    }
27
28    public boolean isDown(int keyCode){
29       return down[keyCode];
30    }
31
32    public boolean isPressed(int keyCode){
33       boolean p = pressed[keyCode];
34       pressed[keyCode] = false;
35       return p;
36    }
37 }
```

6.5 Description

```
1
2  import javax.swing.*;
3  import java.awt.*;
4  import java.awt.event.ActionEvent;
5  import java.awt.event.ActionListener;
6
7  class Description extends JPanel {
8
9
10    public Description() {
11
12       // Set color of menu panel background
13       setBackground(Color.BLACK);
14
15       // Make a box layout of vertical buttons, rigid
             area dimensions determine it's location on
             the y-axis
16       setLayout(new BoxLayout(this, BoxLayout.Y_AXIS))
             ;
17       add(Box.createRigidArea(new Dimension(0, 200)));
```

```
18
19
20          JLabel text = new JLabel("<html>This is a
                parcours game.<br /><br />The player can be
                controlled with the arrow keys"+
21              " or with 'a', 'w' and 'd'.<br />'f' is used
                    to interact with items.<br />The goal is
                    to reach the end"+
22              " (the red rectangle) without dying.<br />To
                    be able to reach this, you first need the
                    key"+
23              " to open the door.<br /><br /></html>"
24          );
25          add(text);
26
27      }
28
29  }
```

6.6 Door

```
1   /*
2    * The door is a placable block that can be opened if
         the player has a key item in the inventory
3    * If the door is closed, it is solid. If it is open,
         it is not.
4    * A door is closed by default
5    */
6
7
8   class Door extends Placable{
9
10      private boolean open = false;
11      private Sprite closedSprite = new Sprite(0xcc6600
            ,0,0,1,2);
12      private Sprite openSprite = new Sprite(0xcc66600
            ,0.1,0,0.2,2);
13
14
15      public Door(Game game){
16          setSize(1,2);
17      }
18
19      public boolean interactable(){
20          return true;
21      }
22
23      public void interact(Actor player){ // open or
            close the door
```

```
24          if (player.hasItem("key"))
25              open = !open;
26      }
27
28      public boolean isSolid(){
29          return !open;
30      }
31
32      public Sprite getSprite(){ // toggle between the
            sprite of an open and the sprite of a closed
            door
33          return open ? openSprite : closedSprite;
34      }
35  }
```

6.7 Exit

```
1
2   /*
3    * The Exit block is a placable block.
4    * If an object interacts with this block, the player
       has won the game
5    */
6   class Exit extends Placable{
7       private Game game;
8       private Sprite sprite = new Sprite(0xff0000
            ,0,0,1,2);
9
10
11      public Exit(Game game){
12          setSize(1,2);
13          this.game = game;
14      }
15
16      public boolean interactable(){
17          return true;
18      }
19
20      public void interact(Actor obj){
21          //System.out.println("exit");
22          game.end(true);
23      }
24
25      public boolean isSolid(){
26          return false;
27      }
28
29      public Sprite getSprite(){
30          return sprite;
```

```
31      }
32  }
```

6.8 Game

```
1   import java.util.*;
2
3   /*
4    * The game class is the main model that is observed
         by the view.
5    * Through the CollisionManager it can keep track of
         all the objects that are placed in the map.
6    * It is also used to load the map data. The method
         load reads all values from the MapLoader and makes
7    * the correct object in the correct place.
8    * The timer will call the step method of the game 30
         times per second. This will call the step method
9    * of the stepManager, which notifies all objects in
         the game with the inputController as argument.
10   * Also, there is a main object, which is a Player.
11   * If the main object dies, the game is lost.
12   * The view can use this main object to keep it in the
          middle of the screen.
13   *
14   */
15  class Game extends Observable {
16
17      private int width, height;
18      private InputController inputData;
19      private CollisionManager collisions;
20      private int centerX, centerY;
21      private Player mainObject;
22      private int time = 50000;
23      private boolean running, won = false, paused =
            false;
24      private Timer timer = new Timer();
25      private StepManager stepManager = new StepManager()
            ;
26
27      private class StepManager extends Observable {
28          private InputController inputController;
29
30          public void setInput(InputController input){
31              inputController = input;
32          }
33
34          public void step(){
35              setChanged();
36              notifyObservers(inputController);
```

```
37          }
38      }
39
40
41      private TimerTask currentTask = new TimerTask(){
42          public void run(){
43              Game.this.step();
44          }
45      };
46
47
48
49      public Set<Placable> getAllObjects(){
50          return collisions.get(0,0,width,height);
51      }
52
53      public Set<Placable> getVisibleObjects(double x,
            double y, double width, double height){
54          return collisions.get(x,y,width,height);
55      }
56
57      public void step(){
58          time--;
59          stepManager.step();
60          setChanged();
61          notifyObservers();
62          if (this.mainObject.isDead() || time<=0){
63              end(false);
64          }
65      }
66
67
68      public void setInputData(InputController inputData)
            {
69          stepManager.setInput(inputData);
70          //this.inputData=inputData;
71      }
72
73      public void setMainObject(Player obj){
74          mainObject = obj;
75      }
76
77      public Player getMainObject(){
78          return mainObject;
79      }
80
81      public int getWidth(){
82          return this.width;
83      }
84
```

```
85      public int getHeight(){
86          return this.height;
87      }
88
89      public int getTime(){
90          return time;
91      }
92
93      public boolean isRunning(){
94          return running;
95      }
96
97      public boolean paused(){
98          return paused;
99      }
100
101     public void end(boolean hasWon){
102         won = hasWon;
103         stop();
104     }
105
106     public boolean finished(){
107         return !running && !paused;
108     }
109
110     public boolean hasWon(){
111         return won;
112     }
113
114     public int getScore(){
115         return won ? mainObject.getHealth()*100+time :
                0;
116     }
117
118     public void start(){
119         running = true;
120         timer.schedule(currentTask,0,33);
121     }
122
123     public void stop(){
124         running = false;
125         setChanged();
126         notifyObservers(null);
127         currentTask.cancel();
128     }
129
130     public void load(String fileName){
131         MapLoader loader = new MapLoader(fileName);
132
133         width=loader.getWidth();
```

```
134         height=loader.getHeight();
135 
136         collisions = new CollisionManager(width,height);
137 
138         for (int x = 0; x<width; x++){
139            for (int y = 0; y<height; y++){
140               Placable obj = getObjectFromInt(loader.get
                     (x,y));
141               if (obj != null){
142                  obj.place(x,y);
143                  obj.putInMap(collisions);
144                  stepManager.addObserver(obj);
145               }
146            }
147         }
148      }
149 
150      public Placable getObjectFromInt(int number){
151         switch(number){
152            case 0xff00ff00:
153               return new Grass(this);
154            case 0xffaaaaaa:
155               return new Stop(this);
156            case 0xff0000ff:
157               return new Player(this);
158            case 0xffff0000:
159               return new Monster(this);
160            case 0xffcc6600:
161               return new Door(this);
162            case 0xffffff00:
163               return new KeyObject(this);
164            case 0xffff00ff:
165               return new Exit(this);
166            default:
167               return null;
168         }
169      }
170 }
```

6.9 GamePanel

```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.util.Collection;
4  import java.util.Timer;
5  import java.util.TimerTask;
6  import java.util.Observer;
7  import java.util.Observable;
8  import java.awt.event.KeyListener;
```

```
9  import java.awt.event.KeyEvent;
10
11 /*
12  * The GamePanel does all the drawing of the game.
13  */
14
15 class GamePanel extends JPanel implements Observer {
16    private Game game;
17    private double scale = 32;
18
19    public GamePanel(Game game){
20       this.game=game;
21       repaint();
22       game.addObserver(this);
23
24    }
25
26    public void paintBorder(Graphics g){
27       g.setFont(new Font(Font.DIALOG,Font.PLAIN,18));
28       g.drawString("Health: "+game.getMainObject().
            getHealth()+"  Time left: "+game.getTime()
            *33.0/1000,getWidth()/2-100,30);
29       if (!game.isRunning()){
30          g.setFont(new Font(Font.DIALOG,Font.BOLD,32))
               ;
31          if (game.paused()){
32             g.drawString("Game paused",getWidth()
                  /2-100,getHeight()/2);
33          } if (game.hasWon()){
34             g.drawString("You won, Score: "+(game.
                  getScore()),getWidth()/2-100,getHeight
                  ()/2);
35          } else {
36             g.drawString("You lost",getWidth()/2-100,
                  getHeight()/2);
37          }
38       }
39    }
40
41    public void paintComponent(Graphics gr){
42       Graphics g = gr.create();
43
44       g.clearRect(0,0,getWidth(),getHeight());
45
46       double x = Math.min(Math.max(game.getMainObject
            ().getX()*scale-getWidth()/2,0),game.getWidth
            ()*scale-getWidth());
47       double y = Math.min(Math.max(game.getMainObject
            ().getY()*scale-getHeight()/2,0),game.
            getHeight()*scale-getHeight());
```

```
48          g.translate(-(int)x,-(int)y);
49          Collection<Placable> gameObjects=game.
               getVisibleObjects(x/scale,y/scale,getWidth()/
               scale,getHeight()/scale);
50
51          for (Placable object : gameObjects){
52             Sprite sprite = object.getSprite();
53             if (sprite == null)
54                continue;
55             drawSprite(sprite, g, object.getX(),object.
                  getY());
56          }
57       }
58
59       public void drawSprite(Sprite sprite, Graphics g,
            double x, double y){
60          if (!sprite.image()){
61             g.setColor(sprite.getColor());
62             g.fillRect((int)((x+sprite.getXmin())*scale)
                  ,(int)((y+sprite.getYmin())*scale),(int)(
                  sprite.getWidth()*scale),(int)(sprite.
                  getHeight()*scale));
63          } else {
64             g.setColor(Color.WHITE);
65             g.drawImage(sprite.getImage(),(int)((x+sprite
                  .getXmin())*scale),(int)((y+sprite.getYmin
                  ())*scale),(int)(sprite.getWidth()*scale)
                  ,(int)(sprite.getHeight()*scale),null);
66          }
67       }
68
69       public void update(Observable caller, Object data){
70          repaint();
71       }
72
73    }
```

6.10 GameWindow

```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.util.ArrayList;
4  import java.util.Timer;
5  import java.util.TimerTask;
6  import java.awt.event.KeyListener;
7  import java.awt.event.KeyEvent;
8  import java.awt.event.ActionEvent;
9  import java.awt.event.ActionListener;
10
```

```
11
12 class GameFrame extends JFrame {
13
14    private JPanel gamePanel, menuPanel, Description;
15    private CardLayout cardLayout = new CardLayout();
16    // A seperate JPanel that holds the game and menu
          cards
17    private JPanel cardContainer;
18    private Game game;
19
20    public GameFrame(Game game, int width, int height){
21       int w = 300, h = 75;
22       JTextField title = new JTextField("Platformer
             Parcours");
23       setTitle(title.getText());
24
25       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
26       setVisible(true);
27       setSize(width,height);
28
29       gamePanel = new GamePanel(game);
30       gamePanel.setSize(width,height);
31
32       menuPanel = new MenuPanel();
33
34       JPanel description = new Description();
35
36
37
38
39       JButton startGame = new JButton("Start Game");
40       startGame.setAlignmentX( Component.
             CENTER_ALIGNMENT );
41       startGame.setMaximumSize( new Dimension(w, h) );
42       startGame.setBackground(Color.GRAY);
43       startGame.setForeground(Color.WHITE);
44       startGame.setBorder(BorderFactory.
             createLineBorder(Color.WHITE, 3));
45       menuPanel.add(startGame);
46       startGame.addActionListener(new ActionListener()
             {
47          public void actionPerformed(ActionEvent e){
48             cardLayout.show(cardContainer,"gamepanel")
                   ;
49             GameFrame.this.game.start();
50             repaint();
51          }
52       });
53
54       JButton howToPlay = new JButton("How to play");
```

```
55          howToPlay.setAlignmentX( Component.
               CENTER_ALIGNMENT );
56          howToPlay.setMaximumSize( new Dimension(w, h) );
57          howToPlay.setBackground(Color.GRAY);
58          howToPlay.setForeground(Color.WHITE);
59          howToPlay.setBorder(BorderFactory.
               createLineBorder(Color.WHITE, 3));
60          menuPanel.add(howToPlay);
61          howToPlay.addActionListener(new ActionListener()
                {
62             public void actionPerformed(ActionEvent e){
63                cardLayout.show(cardContainer,"description
                      ");
64             }
65          });
66
67          JButton quitGame = new JButton("Quit to desktop"
               );
68          quitGame.setAlignmentX( Component.
               CENTER_ALIGNMENT );
69          quitGame.setMaximumSize( new Dimension(w, h) );
70          quitGame.setBackground(Color.GRAY);
71          quitGame.setForeground(Color.WHITE);
72          quitGame.setBorder(BorderFactory.
               createLineBorder(Color.WHITE, 3));
73          menuPanel.add(quitGame);
74          quitGame.addActionListener(new ActionListener()
               {
75             public void actionPerformed(ActionEvent e){
76                System.exit(0);
77             }
78          });
79
80          JButton backToMenu = new JButton("Back to menu")
               ;
81          backToMenu.setAlignmentX( Component.
               CENTER_ALIGNMENT );
82          backToMenu.setMaximumSize( new Dimension(300,
               75) );
83          backToMenu.setBackground(Color.GRAY);
84          backToMenu.setForeground(Color.WHITE);
85          backToMenu.setBorder(BorderFactory.
               createLineBorder(Color.WHITE, 3));
86          description.add(backToMenu);
87          backToMenu.addActionListener(new ActionListener
               (){
88             public void actionPerformed(ActionEvent e){
89                cardLayout.show(cardContainer,"menupanel")
                      ;
90             }
```

```
91          });
92
93
94          cardContainer = new JPanel(cardLayout);
95          cardLayout.addLayoutComponent(gamePanel,"
               gamepanel");
96          cardLayout.addLayoutComponent(menuPanel,"
               menupanel");
97          cardLayout.addLayoutComponent(description,"
               description");
98          cardContainer.add(gamePanel);
99          cardContainer.add(menuPanel);
100         cardContainer.add(description);
101         cardLayout.show(cardContainer,"menupanel");
102
103         add(cardContainer);
104
105         this.game = game;
106
107         InputController controller = new InputController
               ();
108         addKeyListener(controller);
109         game.setInputData(controller);
110
111
112     }
113
114     public GameFrame(Game game){
115         this(game,1200,720);
116     }
117
118
119 }
```

6.11 Grass

```
1  /*
2   * Grass is a normal solid block
3   */
4
5  class Grass extends Block {
6      private Sprite sprite = new Sprite("images/grass.
           png",0,0,1,1);
7
8      public Grass(Game game){
9          setSize(1,1);
10     }
11
12     public Sprite getSprite(){
```

```
13          return sprite;
14      }
15  }
```

6.12 Inventory

```
1   import java.util.ArrayList;
2
3   /*
4    * The inventory can store Items.
5    * There is a maximum capacity of Items in the
         inventory.
6    * Items of the same type are not stackable
7    *
8    * The add method tries to put an item in the
         inventory, and returns whether this was possible.
9    * It is not possible if the item is null, or if the
         inventory is full
10   */
11
12  class Inventory {
13      ArrayList<Item> items = new ArrayList<>();
14      int maxSize;
15
16      public Inventory(int size){
17
18          maxSize = size;
19      }
20
21      public boolean add(Item item){
22          if (items.size()<maxSize && item != null){
23              items.add(item);
24              return true;
25          }
26          return false;
27      }
28
29      public Item remove(Item item){
30          items.remove(item);
31          return item;
32      }
33
34      public Item remove(int index){
35          return items.remove(index);
36      }
37
38      public Item remove(String itemType){
39          for (Item item : items){
40              if (item.getType() == itemType)
```

```
41              items.remove(item);
42              return item;
43          }
44          return null;
45      }
46
47      public Item get(int index){
48          return items.get(index);
49      }
50
51      public boolean contains(Item item){
52          return items.contains(item);
53      }
54
55      public boolean contains(String itemType){
56          for (Item item : items){
57              if (item.getType() == itemType)
58                  return true;
59          }
60          return false;
61      }
62
63
64  }
```

6.13 Item

```
1
2   /*
3    * An Item has a getType method, that returns what
         kind of item it is
4    */
5
6   interface Item{
7
8       public String getType();
9   }
```

6.14 Key

```
1
2   /*
3    * A KeyObject is a placable object that can be picked
         up.
4    * If it is picked up, an item that has "key" as type
         will be given to the object that picked it up/
5    *
```

```
 */

class KeyObject extends Placable {

   Sprite sprite = new Sprite("images/key.png"
      ,0,0,1,1);

   public KeyObject(Game game){
      setSize(1,1);
   }

   public boolean interactable(){
      return true;
   }

   public void interact(Actor player){
      boolean taken = player.take(new Item(){
         public String getType(){
            return "key";
         }
      });
      if (taken){
         removeFromMap();
      }
   }

   public boolean isSolid(){
      return false;
   }

   public Sprite getSprite(){
      return sprite;
   }
}
```

6.15 Main / Parcours

```


class parcours {

   /*
    * To do:
    *
    * make it possible to pause the game
    * draw the inventory
    *
    *
    */
```

```
13
14    public static void main(String[] args){
15       Game game = new Game();
16       if (args.length>0){
17          game.load(args[0]);
18       } else {
19          game.load("map.png");
20       }
21
22       new GameFrame(game);
23    }
24 }
```

6.16 Maploader

```
1  import java.io.*;
2  import java.awt.image.*;
3  import javax.imageio.*;
4
5  /*
6   * The MapLoader loads an image file.
7   * It will can then return the color and transparency
       value of a pixel as an integer
8   *
9   */
10
11 class MapLoader {
12
13    private BufferedImage img = null;
14    private int width, height;
15
16    public MapLoader(String fileName){
17
18       load(fileName);
19    }
20
21    public void load(String fileName){
22
23       try {
24          img = ImageIO.read(new File(fileName));
25          width = img.getWidth();
26          height = img.getHeight();
27       } catch (FileNotFoundException e){
28          System.err.println("Unable to open file: '"+
               fileName+"': "+e.getMessage());
29       } catch (IOException e) {
30          e.printStackTrace();
31       }
32
```

```
33      }
34
35      public int get(int x, int y){
36          return img.getRGB(x,y);
37      }
38
39      public int getWidth(){
40          return width;
41      }
42
43      public int getHeight(){
44          return height;
45      }
46
47      public void print(){
48          String output = "";
49          for (int y=0; y<height; y++){
50              for (int x=0; x<width; x++){
51                  output += Integer.toHexString(get(x,y))+"
                        ";
52              }
53              output += "\n";
54          }
55          System.out.println(output);
56      }
57
58  }
```

6.17 Menupanel

```
1   import javax.swing.*;
2   import java.awt.*;
3   import java.awt.event.ActionEvent;
4   import java.awt.event.ActionListener;
5
6   class MenuPanel extends JPanel {
7       // Dimensions of the button
8
9       public MenuPanel() {
10
11          // Set color of menu panel background
12          setBackground(Color.BLACK);
13
14          // Make a box layout of vertical buttons, rigid
                area dimensions determine it's location on
                the y-axis
15          setLayout(new BoxLayout(this, BoxLayout.Y_AXIS))
                ;
16          add(Box.createRigidArea(new Dimension(0, 200)));
```

```
17
18
19      }
20  }
```

6.18 Monster

```
1   import java.util.Observer;
2   import java.util.Observable;
3
4   /*
5    * A monster is a movable object, and has therefore
         all pyisics working on it.
6    * A monster does 8 damage to colliding objects that
         can be damaged
7    * It's AI can be described as:
8    *     move in the chosen direction (in the beginning
         left by default)
9    *     if there is a block or a gap in front: turn
         around
10   */
11
12  class Monster extends Movable {
13
14      private boolean direction = false;
15      private Sprite sprite = new Sprite("images/monster.
            png",0,0,1,1);
16
17      public Monster(Game game){
18
19          setSize(1,1);
20      }
21
22      public Sprite getSprite(){
23          return sprite;
24      }
25
26      public boolean left(){
27          return !direction;
28      }
29      public boolean right(){
30          return direction;
31      }
32      public boolean jump(){
33          return false;
34      }
35      public boolean fly(){
36          return false;
37      }
```

```
38
39      public boolean isSolid(){
40          return false;
41      }
42
43      public void update(Observable caller, Object data){
44          if (data == null)
45              return;
46
47          updateCollisions();
48
49          updatePhysics();
50          if (NextToSolidObject(direction ? 0.1 :
                -0.1,0.0) || !NextToSolidObject(direction ?
                0.5 : -0.5,0.2)){ // turn around if it is
                next to a block or gap
51              direction = !direction;
52          }
53      }
54
55      private boolean NextToSolidObject(double x, double
            y){
56          move(x,y);
57          boolean collision = solidCollision()!=null;
58          move(-x,-y);
59          return collision;
60      }
61
62      public int damage(){
63          return 8;
64      }
65
66      public double getWalkForce(){
67          return 0.08;
68      }
69      public double getJumpForce(){
70          return 0.0;
71      }
72      public double getGroundFriction(){
73          return 0.6;
74      }
75      public double getAirFrictionX(){
76          return 0.9;
77      }
78      public double getAirFrictionY(){
79          return 0.988;
80      }
81      public double getGravity(){
82          return 0.016;
83      }
```

```
84
85 }
```

6.19 Movable

```
1  import java.util.Set;
2  import java.util.HashSet;
3
4  /*
5   * Any object that inherits from Movable will be
        stopped by solid objects.
6   * These objects also have a gravity and friction on
        both the x and y axis.
7   * Furthermore, these objects can move left, right and
         jump
8   * The methods for going left, going right, jumping
        and flying return whether the object wants to take
         these actions
9   * The UpdatePhysics method will try to make this
        happen
10  * Flying means jumping while in the air
11  *
12  * To test where it can move collision-free, the
        object has some methods to get the colliding
        objects
13  */
14
15 abstract class Movable extends Placable {
16
17
18    private double xSpeed = 0;
19    private double ySpeed = 0;
20    private boolean onground = false;
21
22    private Set<Placable> closeObjects = new HashSet<
         Placable>();
23
24    public abstract double getWalkForce();
25    public abstract double getJumpForce();
26    public abstract double getGroundFriction();
27    public abstract double getAirFrictionX(); // the
         horizontal air frictionhas not been used.
         Currently, the ground friction is used for this.
28    public abstract double getAirFrictionY();
29    public abstract double getGravity();
30
31    public abstract boolean left();
32    public abstract boolean right();
33    public abstract boolean jump();
```

```
34      public abstract boolean fly();
35
36
37      public void updateCollisions(){
38          closeObjects = getCollisionManager().get(getX()
                -1,getY()-1,getWidth()+2,getHeight()+2);
39      }
40
41      public Set<Placable> getCloseObjects(){
42          return closeObjects;
43      }
44
45      public Placable solidCollision(){
46          for (Placable o : closeObjects){
47              if (this.collidesWith(o) && o.isSolid() &&
                    this != o){
48                  return o;
49              }
50          }
51          return null;
52      }
53
54      public void updatePhysics(){
55
56          xSpeed += (right() ? getWalkForce() : 0.0) - (
                left() ? getWalkForce() : 0.0);
57          xSpeed *= getGroundFriction();
58
59          move(xSpeed,0); // try to move on the x axis
60          Placable o = solidCollision();
61          if (o != null){ // if blocked, go to the closest
                free position
62              if (xSpeed>0)
63                  place(o.getX() - getWidth(), getY());
64              else
65                  place(o.getX() + o.getWidth(), getY());
66          }
67
68
69          ySpeed += getGravity();
70          ySpeed *= getAirFrictionY();
71          ySpeed = ((jump() && onground) || fly()) ? -
                getJumpForce() : ySpeed;
72
73          onground = false;
74
75          move(0,ySpeed); // try to move on the y axis
76          o = solidCollision();
77          if (o != null){ // if blocked, go to the closest
                free position
```

```
78          if (ySpeed>0){
79             place(getX(), o.getY()-getHeight());
80             onground = true;
81          } else
82             place(getX(),o.getY()+o.getHeight());
83          ySpeed = 0;
84       }
85
86
87    }
88 }
```

6.20 Placable

```
1  import java.util.Observer;
2  import java.util.Observable;
3
4  /*
5   * Every object that is placed in the map is a
        Placable.
6   * A placable object has an x and y position, a width
        and height, a method telling whether it can
        interact,
7   * and if so, it should overwrite the interact method,
        a method telling whether it is solid.
8   *
9   * All placable objects observe the step manager of
        the game object.
10  */
11
12
13 abstract class Placable implements Observer{
14
15    private double x, y;
16    private double w, h;
17    private CollisionManager collisionManager = null;
18
19
20    public void putInMap(CollisionManager collisions){
21       collisionManager = collisions;
22       collisionManager.set(this);
23    }
24
25    public double getX(){
26       return x;
27    }
28
29    public double getY(){
30       return y;
```

```
31      }
32
33      public double getWidth(){
34          return w;
35      }
36
37      public double getHeight(){
38          return h;
39      }
40
41      public void place(double x, double y){
42          if (collisionManager != null)
43              collisionManager.remove(this);
44          this.x = x;
45          this.y = y;
46          if (collisionManager != null)
47              collisionManager.set(this);
48      }
49
50      protected void setSize(double width, double height)
            {
51          if (collisionManager != null)
52              collisionManager.remove(this);
53          w = width;
54          h = height;
55          if (collisionManager != null)
56              collisionManager.set(this);
57      }
58
59      public void move(double x, double y){
60          if (collisionManager != null)
61              collisionManager.remove(this);
62          this.x += x;
63          this.y += y;
64          if (collisionManager != null)
65              collisionManager.set(this);
66      }
67
68      public void removeFromMap(){
69          collisionManager.remove(this);
70      }
71
72      public CollisionManager getCollisionManager(){
73          return collisionManager;
74      }
75
76      public boolean collidesWith(Placable obj){
77          double l1 = getX(), r1=getX()+getWidth(), t1 =
                getY(), b1 = getY()+getHeight();
```

```
78        double l2 = obj.getX(), r2 = obj.getX()+obj.
              getWidth(), t2 = obj.getY(), b2 = obj.getY()+
              obj.getHeight();
79        return !( l1>=r2 || l2>=r1 || t1>=b2 || t2>=b1 )
              ;
80    }
81
82    public abstract Sprite getSprite();
83
84    public abstract boolean isSolid();
85
86    public boolean interactable(){
87        return false;
88    }
89
90    public void update(Observable caller, Object data)
          {}
91
92    public void interact(Actor obj){}
93
94    public int damage(){
95        return 0;
96    }
97
98 }
```

6.21 Player

```
1  import java.awt.event.KeyListener;
2  import java.awt.event.KeyEvent;
3  import java.util.Observer;
4  import java.util.Observable;
5
6  /*
7   * The player is the main object in the game.
8   * It is controlled by the left, up and right arrow
        keys, or by the A, W and D keys.
9   * F can be used to interact with objects.
10  * The spacebar is a kind of cheat key that allows the
        player to jump while in the air.
11  * The player has an invertory, and some methods from
        the interface Actor that modify or describe this
        inventory.
12  *
13  * Sprite creator: Hobojoe0858 on deviantart
14  * Source: http://hobojoe0858.deviantart.com/art/
        Gordon-Freeman-Sprite-136080175
15  */
16
```

```
17 class Player extends Movable implements Actor{
18
19    private Sprite sprite = new Sprite("images/
          Gordon_Freeman_Sprite_by_HoboJoe0858.png"
          ,0,0,1,2);
20    private int health = 100;
21    private int score = 0;
22
23
24    private boolean goLeft, goRight, goJump, goFly,
          interact;
25    private Inventory inventory = new Inventory(12);
26
27
28
29    public Player(Game game){
30       setSize(1,2);
31       game.setMainObject(this);
32    }
33
34
35
36
37    public void update(Observable caller, Object data){
38       if (data == null)
39          return;
40       InputController inputData = (InputController)
             data;
41
42       updateCollisions();
43
44       for (Placable object : getCloseObjects() ){
45          if ( this.collidesWith(object) ){
46             this.health -= object.damage();
47          }
48       }
49       goLeft = inputData.isDown(65) || inputData.
             isDown(37);
50       goRight = inputData.isDown(68) || inputData.
             isDown(39);
51       goJump = inputData.isDown(87) || inputData.
             isDown(38);
52       goFly = inputData.isDown(32); // actually
             cheating
53       interact = inputData.isPressed(70); // interact
             with near objects if you press the 'f' key
54
55
56
57       updatePhysics();
```

```
58
59
60          if (interact){ // interactions can be things
                like opening a door, picking up something or
                pushing a button
61              for (Placable obj : getCloseObjects()){
62                  if (obj.interactable() && closeTo(obj,0.2)
                        )
63                      obj.interact(this);
64              }
65          }
66
67      }
68
69      private boolean closeTo(Placable obj, double
            maxDist){ // return whether the objet is within
            a distance of maxDist of the player
70          // this means that if the player would move
                maxdist to the object they would collide
71          move(-maxDist,-maxDist);
72          setSize(getWidth()+maxDist*2, getHeight()+
                maxDist*2);
73          boolean close = collidesWith(obj);
74          setSize(getWidth()-maxDist*2, getHeight()-
                maxDist*2);
75          move(maxDist,maxDist);
76          return close;
77      }
78
79      public boolean take(Item item){
80          return inventory.add(item);
81      }
82
83      public boolean hasItem(String itemType){
84          return inventory.contains(itemType);
85      }
86
87      public Item loseItem(String itemType){
88          return inventory.remove(itemType);
89      }
90
91      public Inventory getInventory(){
92          return inventory;
93      }
94
95      public Sprite getSprite(){
96          return sprite;
97      }
98
99      public int getHealth(){
```

```
100         return health;
101     }
102 
103     public void setHealth(int newHealth){
104         health = newHealth;
105     }
106 
107     public boolean isDead(){
108         return health<0;
109     }
110 
111     public boolean isSolid(){
112         return false;
113     }
114 
115     public boolean left(){
116         return goLeft;
117     }
118 
119     public boolean right(){
120         return goRight;
121     }
122 
123     public boolean jump(){
124         return goJump;
125     }
126 
127     public boolean fly(){
128         return goFly;
129     }
130 
131     public double getWalkForce(){
132         return 0.1;
133     }
134     public double getJumpForce(){
135         return 0.5;
136     }
137     public double getGroundFriction(){
138         return 0.6;
139     }
140     public double getAirFrictionX(){
141         return 0.9;
142     }
143     public double getAirFrictionY(){
144         return 0.988;
145     }
146     public double getGravity(){
147         return 0.016;
148     }
149 
```

```
150
151 }
```

6.22 Sprite

```
1  import java.awt.Color;
2  import java.awt.Graphics;
3  import java.io.*;
4  import java.awt.image.*;
5  import javax.imageio.*;
6
7  /*
8   * A Sprite object can either be an image that is
        drawn somwhere on the screen, or a colored
        rectangle.
9   * Which of the two depents on the first argument to
        the constructor: if it is an integer, a color
10  * will be made from this integer, and if it is a
        string, the file with that name is used as image.
11  */
12
13 class Sprite {
14
15    private double xmin, ymin, width, height;
16    private Color color;
17    private BufferedImage img;
18    private boolean hasImg = false;
19
20    public Sprite(int color, double xmin, double ymin,
          double width, double height){
21       // in this case it will just be a colored
             rectangle
22       this.color=new Color(color);
23       this.xmin=xmin;
24       this.ymin=ymin;
25       this.width=width;
26       this.height=height;
27    }
28
29    public Sprite(String fileName,double xmin, double
          ymin, double width, double height){
30       // in this case, an image will actually be drawn
31       this.xmin = xmin;
32       this.ymin = ymin;
33       this.width = width;
34       this.height = height;
35
36       try {
37          img = ImageIO.read(new File(fileName));
```

```
38            width = img.getWidth();
39            height = img.getHeight();
40        } catch (IOException e) {
41            System.err.println("Error, couldn't load file
                  '"+fileName+"': "+e);
42        }
43
44        hasImg = true;
45
46
47    }
48
49    public double getXmin(){
50        return xmin;
51    }
52    public double getYmin(){
53        return ymin;
54    }
55
56    public double getWidth(){
57        return width;
58    }
59    public double getHeight(){
60        return height;
61    }
62
63    public boolean image(){
64        return hasImg;
65    }
66
67    public Color getColor(){
68        return color;
69    }
70
71    public BufferedImage getImage(){
72        return img;
73    }
74
75 }
```

6.23 Stopper

```
1
2  /*
3   * This is an invisible placable object that is used
        to stop the player at the edges
4   */
5
6  class Stop extends Block{
```

```
7
8       public Stop(Game game){
9           setSize(1,1);
10      }
11
12      public Sprite getSprite(){
13          return null;
14      }
15  }
```