Dr. TimeBender
Author Batalan Vlad

Gameplay (rules): Single player campaign where the player must advance through different rooms by using a time travel game mechanic which saves the previous moves of the player and creates an old instance of the player that follows that move pattern. The game gamplay involves solving different puzzles in which the player must synchronise with the past versions of himself, in order to activate/deactivate different paths that lead towards the completion of the level. Time travel is dangerous because it can create time paradoxes and weird behavior sometimes, so it must be used carefully.

Plot (game story): Dr. TimeBender has been taken prisoner by his worst enemy: his father — Dr. VoidBender — an adept of the aplications of dark science. Dr. VoidBender locked his son in his most hidden and well-secured laboratory. Dr. TimeBender must escape from his father's laboratory using the power of time travel which grants him the ability to be in saveral places at the same time. He must overcome the lab's security and stop his father from conducting an experiment that threatens to destroy the universe. Time travel can be a dangerous tool though, and it certanly has its drawbacks. It can bring forth time paradoxes or create weird circumstances and behavior patterns. The more doc uses his power, the more time begins to pass slower and slower.

Characters:

i) **Dr. TimeBender** is the protagonist of the game. He is a very smart, patient and calm scientist that developed the time travel. He refuses to contribute to his father's expleriments because they endanger the existence of the universe.

ii) **Dr. TimeBender's old version** is the resulting instance of time travel. From his perspective, his behavior is due to his own will and freedom, but in reality, it is already determined. He is aware of the older versions of himself, but completely blind to the instances that come from the future. This half-aware way to percieve the world is the main factor that creates time paradoxes.

iii) **Dr. VoidBender** is the main antagonist of the story. Dispite the fact that he is absent from the gameplay, he is the main goul of the game. He is a restless scientist who studys the dark science, involving the use of the dark energy with the risk of destroying the universe. Interested only to obtain imense powers, he considers all the living beings in the universe only test subjects to sacrifice in order to discover the true power of dark science.

Mechanics: The main game mechanic is time travel. In a level, there are saveral strategic placed doors that lead to the objective of the game. It can be opened by going to a switch or a lever that activates it. As a player, alone, it is impossible to be next to the switch to maintain the door open and to pass through the door in the same time. The stategy, that is conisdered to be the main theme of the game, is to create a past verson of yourself that holds the door, just enough for you to pass through it.

The game contains also special objects such as moving platforms, controlled moving platforms, scallers that change it's behaviour based on the mass of each of the tilers.

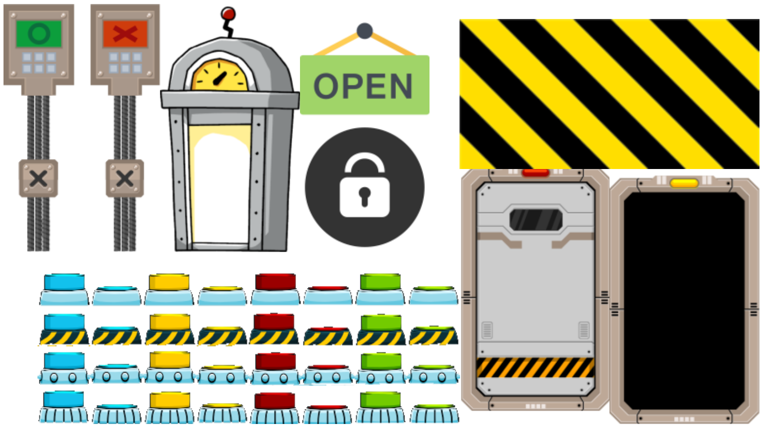
The controlls of the game are very simple. Use the arrow keys (up, left, right) to move the player and do different actions such as time travelling by pressing the space key.

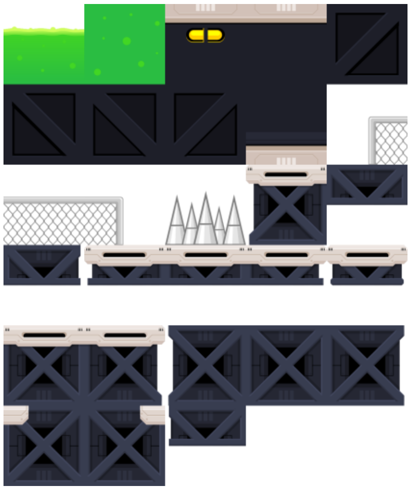
Winning condition: The player must reach the door that leads to the next level and press the Space Key when the objective door is unlocked.

Losing conditions: The game is lost when the player, or one old instance of himself dies does not reach the time travel machine when he tries to teleport in time, leading to a paradox.

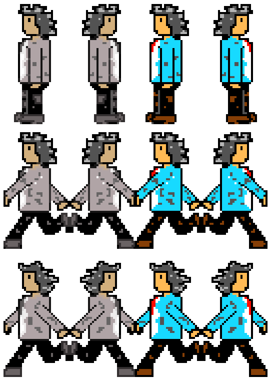
Sprite sheets used:

i) Tiles and objects Spritesheet:

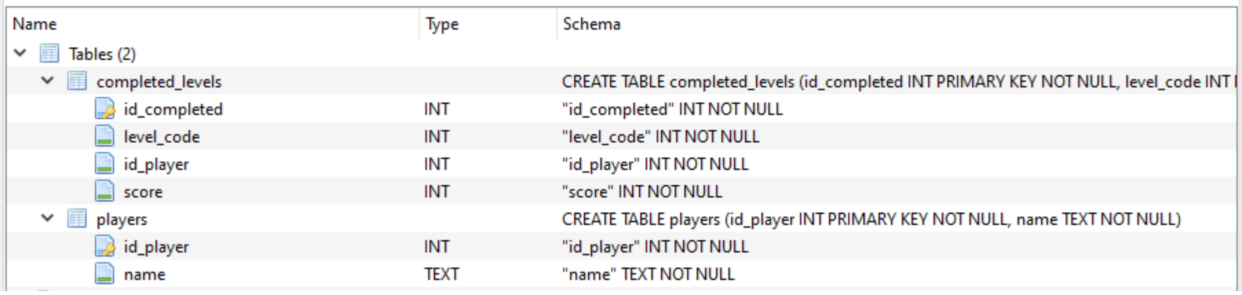




ii) Player Spritesheet:



The Database of the game: It includes two main tables.



Name	Type	Schema
Tables (2)		
completed_levels		CREATE TABLE completed_levels (id_completed INT PRIMARY KEY NOT NULL, level_code INT
id_completed	INT	"id_completed" INT NOT NULL
level_code	INT	"level_code" INT NOT NULL
id_player	INT	"id_player" INT NOT NULL
score	INT	"score" INT NOT NULL
players		CREATE TABLE players (id_player INT PRIMARY KEY NOT NULL, name TEXT NOT NULL)
id_player	INT	"id_player" INT NOT NULL
name	TEXT	"name" TEXT NOT NULL

The first one is called players and stores the profiles with the id number associated.

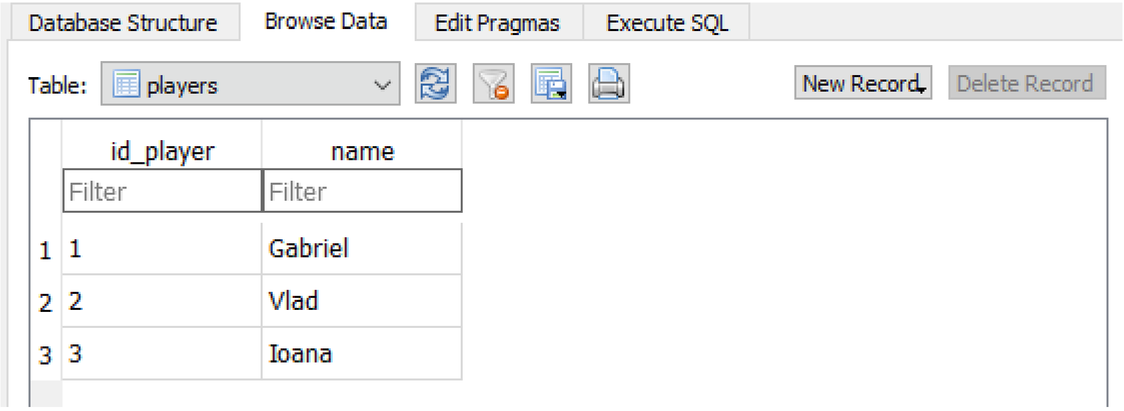


Table: players

	id_player	name
1	1	Gabriel
2	2	Vlad
3	3	Ioana

The second table is called completed_levels and stores the data from a fully completed level, such as the profile that completed the level, the code of the level, and the score obtained exprimated in game ticks.

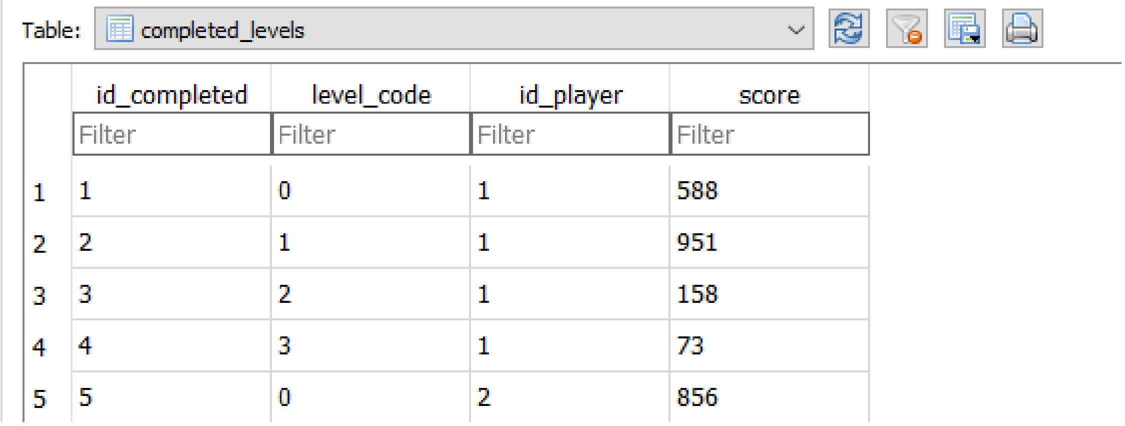


Table: completed_levels

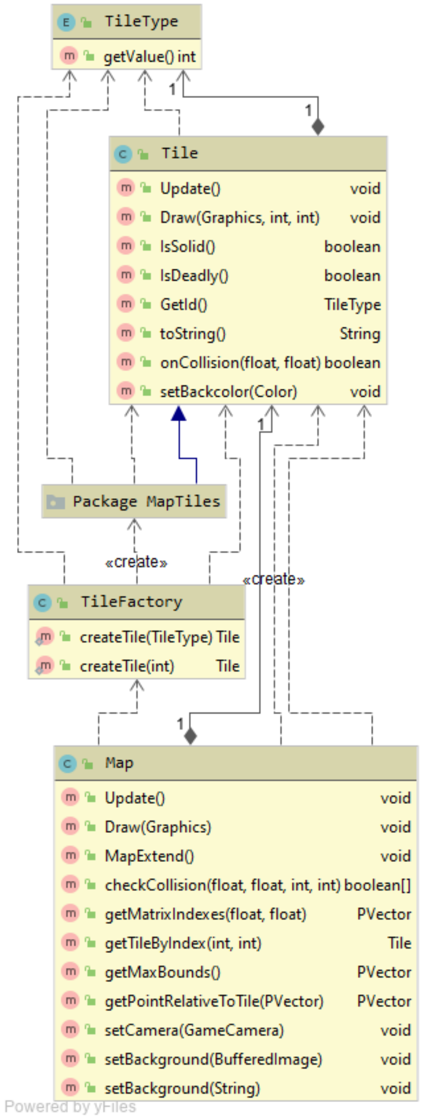
	id_completed	level_code	id_player	score
1	1	0	1	588
2	2	1	1	951
3	3	2	1	158
4	4	3	1	73
5	5	0	2	856

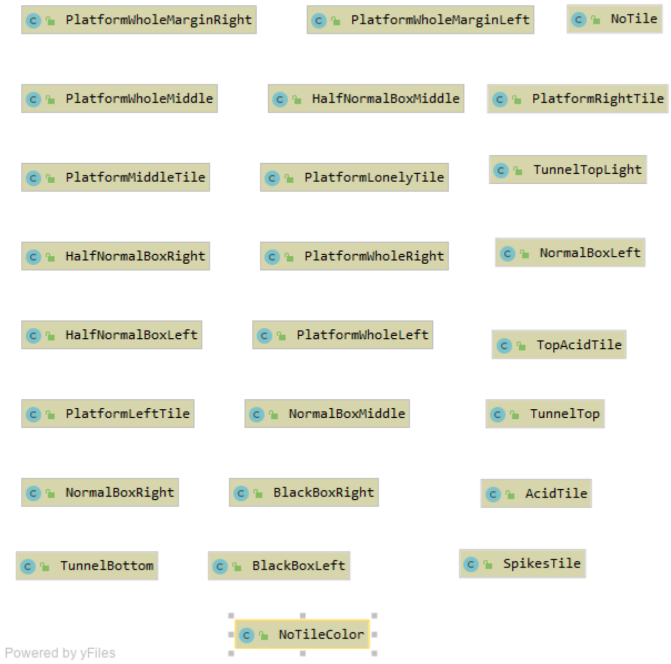
Technical documentation of the clases:

The main components of the architecture of the game are the Map, the Levels, the GameObjects, the collision between the objects and the map, the Database.

i) **The Map** is build from an orthogonal perspective using different Tiles. Each tile is assigned a value such that reading a map dirrectly from a text file is possible.

In order to create each tile based on the specific id of that particulary tile, a Factory design pattern was used.

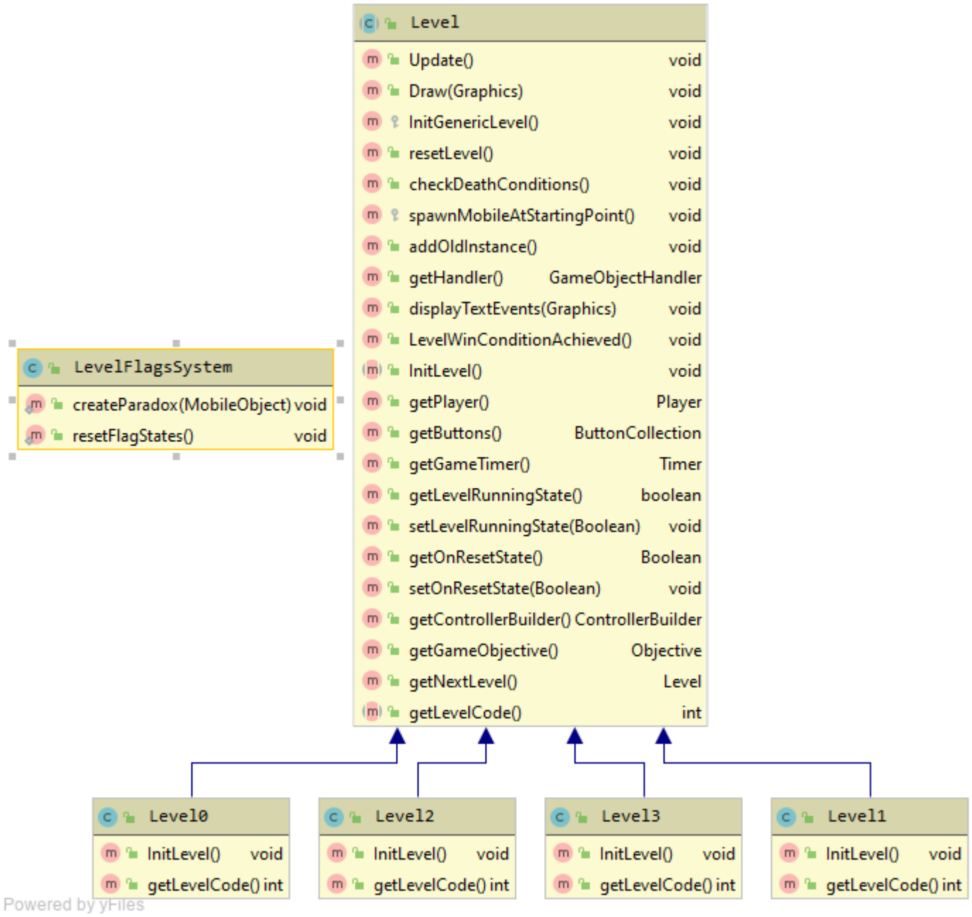

TileType
getValue() int
1
1
Tile
Update() void
Draw(Graphics, int, int) void
IsSolid() boolean
IsDeadly() boolean
GetId() TileType
toString() String
onCollision(float, float) boolean
setBackcolor(Color) void
1
Package MapTiles
«create»
«create»
TileFactory
createTile(TileType) Tile
createTile(int) Tile
1
Map
Update() void
Draw(Graphics) void
MapExtend() void
checkCollision(float, float, int, int) boolean[]
getMatrixIndexes(float, float) PVector
getTileByIndex(int, int) Tile
getMaxBounds() PVector
getPointRelativeToTile(PVector) PVector
setCamera(GameCamera) void
setBackground(BufferedImage) void
setBackground(String) void
Powered by yFiles



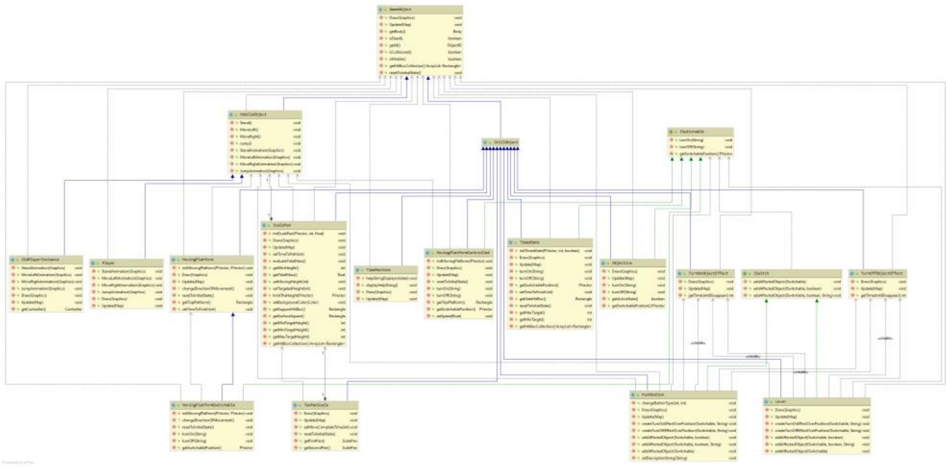
ii) **The Levels** are classes that describe the context of the game. One level has included a map and all the objects that exists. The Level abstract class is responsible for drawing the screen, updating all the objects, for the conditions of winning and the losing, reseting each level to the initial conditions, spawning each object to its original position and time reset.

The concrete level clases (Level0, Level1, …) are only responsible for defining the objects that exist and the map in which the action takes place. Some important flags that describe the state of the level are included in the LevelFlagSystem class.

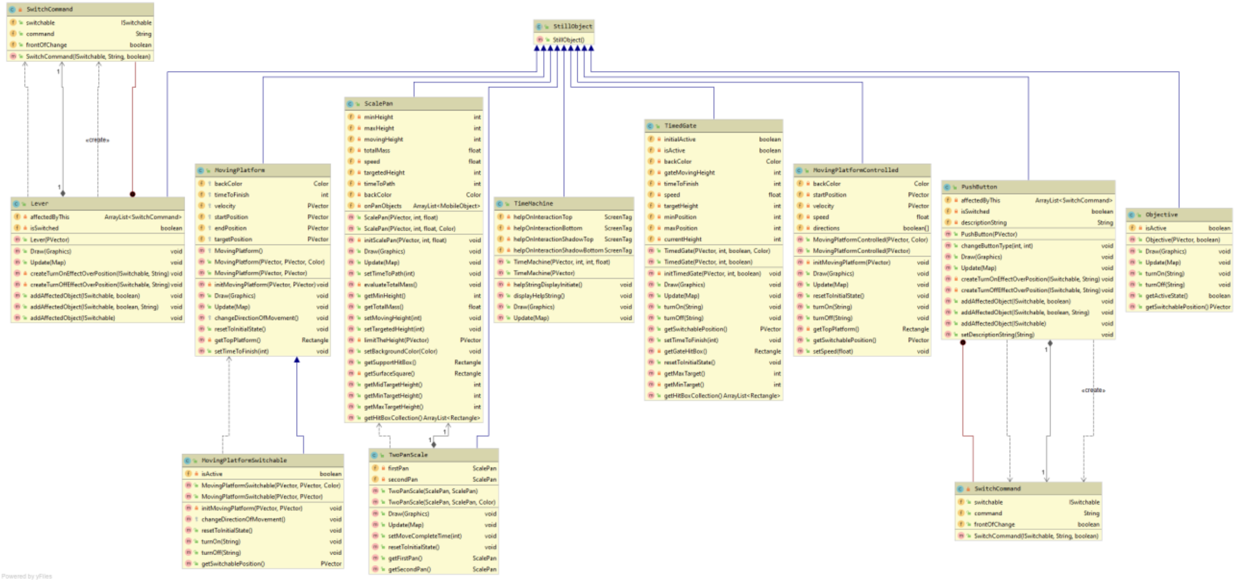
The time taken to finished a level considered to be the score's unit of measurement. The target of the level is to be completed in the least amount of time.



iii) **The Game Objects** are separated into two categories. There are mobile objects, that can use the move left/right, jump commands in order to change position, and still objects that either cannot move at all or can move but through other methods than the mobile objects.



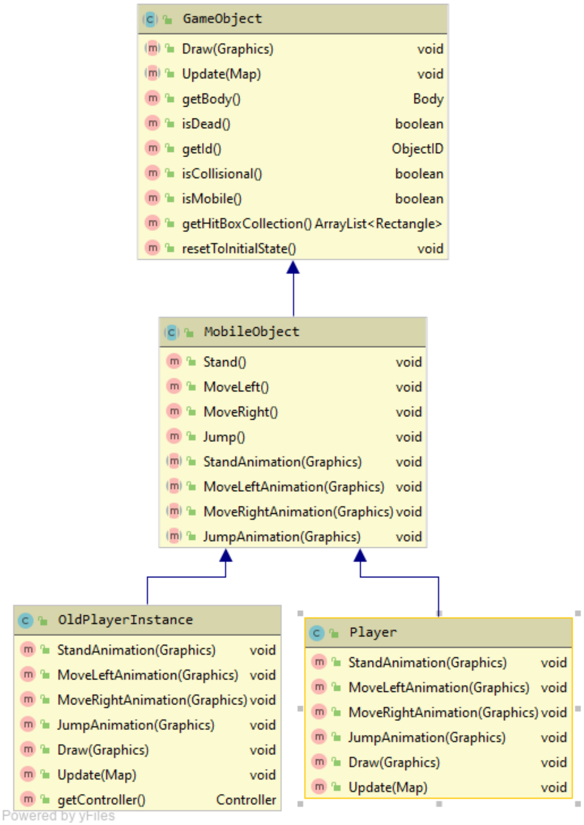
Still objects that implement ISwitchable can be controlled by those that implement the ISwitch interface. One example consists in the TimedGate cobject, that can be turned on or off using a Lever or a PushButton.


SwitchCommand
switchable ISwitchable
command String
frontOfChange boolean
SwitchCommand(ISwitchable, String, boolean)
1
1
«create»
Lever
affectedByThis ArrayList<SwitchCommand>
isSwitched boolean
Lever(PVector)
Draw(Graphics) void
Update(Map) void
createTurnOnEffectOverPosition(ISwitchable, String) void
createTurnOffEffectOverPosition(ISwitchable, String) void
addAffectedObject(ISwitchable, boolean) void
addAffectedObject(ISwitchable, boolean, String) void
addAffectedObject(ISwitchable) void
MovingPlatform
backColor Color
timeToFinish int
velocity PVector
startPosition PVector
endPosition PVector
targetPosition PVector
MovingPlatform()
MovingPlatform(PVector, PVector, Color)
MovingPlatform(PVector, PVector)
initMovingPlatform(PVector, PVector) void
Draw(Graphics) void
Update(Map) void
changeDirectionOfMovement() void
resetToInitialState() void
getTopPlatform() Rectangle
setTimeToFinish(int) void
StillObject
StillObject()
ScalePan
minHeight int
maxHeight int
movingHeight int
totalMass float
speed float
targetedHeight int
timeToPath int
backColor Color
onPanObjects ArrayList<MobileObject>
ScalePan(PVector, int, float)
ScalePan(PVector, int, float, Color)
initScalePan(PVector, int, float) void
Draw(Graphics) void
Update(Map) void
setTimeToPath(int) void
evaluateTotalMass() void
getMinHeight() int
getTotalMass() float
setMovingHeight(int) void
setTargetedHeight(int) void
limitTheHeight(PVector) PVector
setBackgroundColor(Color) void
getSupportHitBox() Rectangle
getSurfaceSquare() Rectangle
getMidTargetHeight() int
getMinTargetHeight() int
getMaxTargetHeight() int
getHitBoxCollection() ArrayList<Rectangle>
TimeMachine
helpOnInteractionTop ScreenTag
helpOnInteractionBottom ScreenTag
helpOnInteractionShadowTop ScreenTag
helpOnInteractionShadowBottom ScreenTag
TimeMachine(PVector, int, int, float)
TimeMachine(PVector)
helpStringDisplayInitiate() void
displayHelpString() void
Draw(Graphics) void
Update(Map) void
TimedGate
initialActive boolean
isActive boolean
backColor Color
gateMovingHeight int
timeToFinish int
speed float
targetHeight int
minPosition int
maxPosition int
currentHeight int
TimedGate(PVector, int, boolean, Color)
TimedGate(PVector, int, boolean)
initTimedGate(PVector, int, boolean) void
Draw(Graphics) void
Update(Map) void
turnOn(String) void
turnOff(String) void
getSwitchablePosition() PVector
setTimeToFinish(int) void
getGateHitBox() Rectangle
resetToInitialState() void
getMaxTarget() int
getMinTarget() int
getHitBoxCollection() ArrayList<Rectangle>
MovingPlatformControlled
backColor Color
startPosition PVector
velocity PVector
speed float
directions boolean[]
MovingPlatformControlled(PVector, Color)
MovingPlatformControlled(PVector)
initMovingPlatform(PVector) void
Draw(Graphics) void
Update(Map) void
resetToInitialState() void
turnOn(String) void
turnOff(String) void
getTopPlatform() Rectangle
getSwitchablePosition() PVector
setSpeed(float) void
PushButton
affectedByThis ArrayList<SwitchCommand>
isSwitched boolean
descriptionString String
PushButton(PVector)
changeButtonType(int, int) void
Draw(Graphics) void
Update(Map) void
createTurnOnEffectOverPosition(ISwitchable, String) void
createTurnOffEffectOverPosition(ISwitchable, String) void
addAffectedObject(ISwitchable, boolean) void
addAffectedObject(ISwitchable, boolean, String) void
addAffectedObject(ISwitchable) void
setDescriptionString(String) void
Objective
isActive boolean
Objective(PVector, boolean)
Draw(Graphics) void
Update(Map) void
turnOn(String) void
turnOff(String) void
getActiveState() boolean
getSwitchablePosition() PVector
1
«create»
1
MovingPlatformSwitchable
isActive boolean
MovingPlatformSwitchable(PVector, PVector, Color)
MovingPlatformSwitchable(PVector, PVector)
initMovingPlatform(PVector, PVector) void
changeDirectionOfMovement() void
resetToInitialState() void
turnOn(String) void
turnOff(String) void
getSwitchablePosition() PVector
1
1
TwoPanScale
firstPan ScalePan
secondPan ScalePan
TwoPanScale(ScalePan, ScalePan)
TwoPanScale(ScalePan, ScalePan, Color)
Draw(Graphics) void
Update(Map) void
setMoveCompleteTime(int) void
resetToInitialState() void
getFirstPan() ScalePan
getSecondPan() ScalePan
1
SwitchCommand
switchable ISwitchable
command String
frontOfChange boolean
SwitchCommand(ISwitchable, String, boolean)
Powered by yFiles

There are two mobile objects. One is the PlayerObject and the other is the OldPlayerInstance.

One important mechanic of the game is represented by the way the previous moves of the Player are passed to the OldPlayerInstance in order to be followed precisely and simulate the past of the Player.

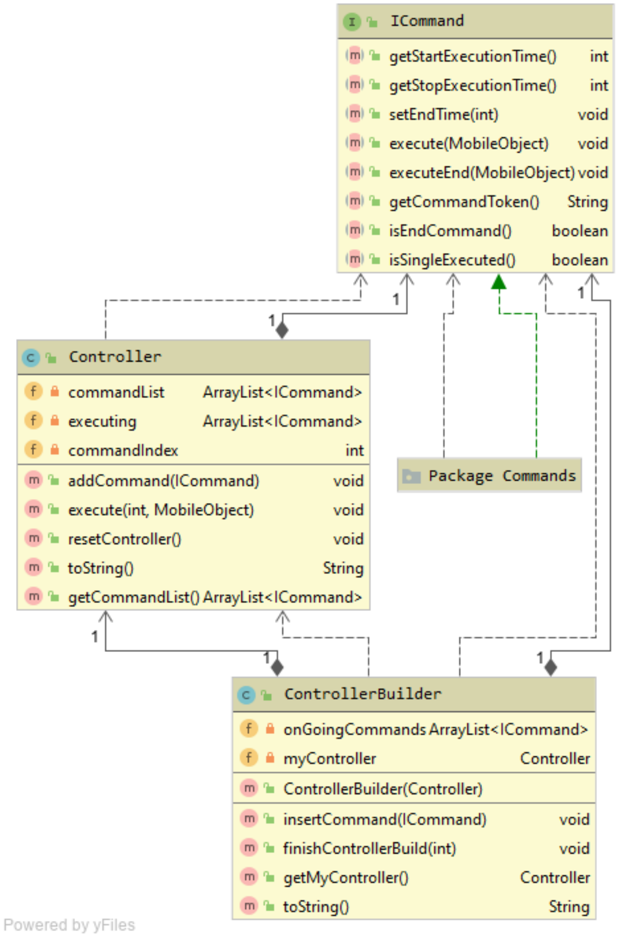
For implementing this feature, I used two design patterns. The first one is the command design pattern which was used to incapsulate the keys pressed by the player and save them

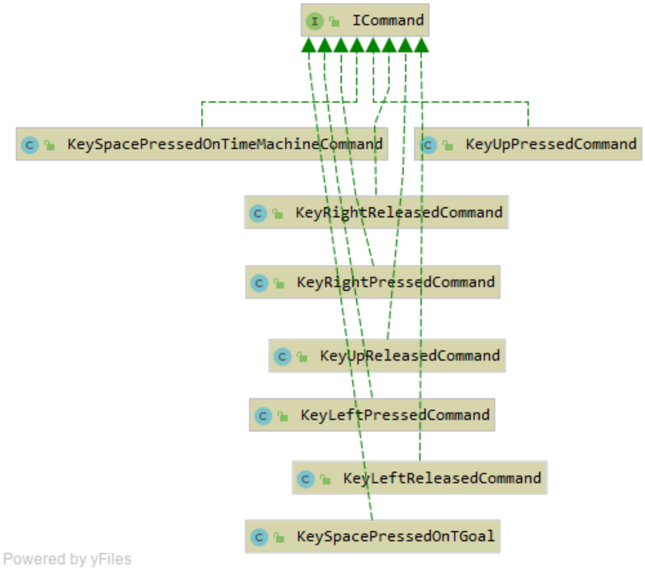

GameObject
Draw(Graphics) void
Update(Map) void
getBody() Body
isDead() boolean
getId() ObjectID
isCollisional() boolean
isMobile() boolean
getHitBoxCollection() ArrayList<Rectangle>
resetToInitialState() void
MobileObject
Stand() void
MoveLeft() void
MoveRight() void
Jump() void
StandAnimation(Graphics) void
MoveLeftAnimation(Graphics) void
MoveRightAnimation(Graphics) void
JumpAnimation(Graphics) void
OldPlayerInstance
StandAnimation(Graphics) void
MoveLeftAnimation(Graphics) void
MoveRightAnimation(Graphics) void
JumpAnimation(Graphics) void
Draw(Graphics) void
Update(Map) void
getController() Controller
Player
StandAnimation(Graphics) void
MoveLeftAnimation(Graphics) void
MoveRightAnimation(Graphics) void
JumpAnimation(Graphics) void
Draw(Graphics) void
Update(Map) void
Powered by yFiles

into an array from the Controller class.

In order to optimize the number of commands in this array, for each command was assign a starting time and an ending time, an executing method and a stop executing method which simulate the efects of the release of the key.

Making this optimization has increased the difficulty of creating a Controller. This is the reason why the second design pattern was used – the Builder pattern. While the game is running, the ControllerBuilder class assures to create the Controller that will be used by the next OldPlayerInstance that will be created.

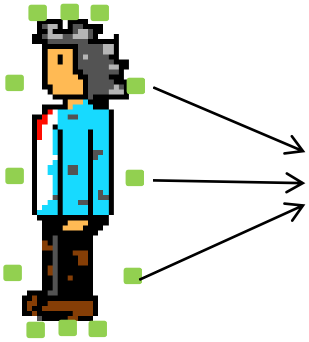

ICommand
getStartExecutionTime() int
getStopExecutionTime() int
setEndTime(int) void
execute(MobileObject) void
executeEnd(MobileObject) void
getCommandToken() String
isEndCommand() boolean
isSingleExecuted() boolean
Controller
commandList ArrayList<ICommand>
executing ArrayList<ICommand>
commandIndex int
addCommand(ICommand) void
execute(int, MobileObject) void
resetController() void
toString() String
getCommandList() ArrayList<ICommand>
Package Commands
ControllerBuilder
onGoingCommands ArrayList<ICommand>
myController Controller
ControllerBuilder(Controller)
insertCommand(ICommand) void
finishControllerBuild(int) void
getMyController() Controller
toString() String
1
1
1
1
1
1
Powered by yFiles



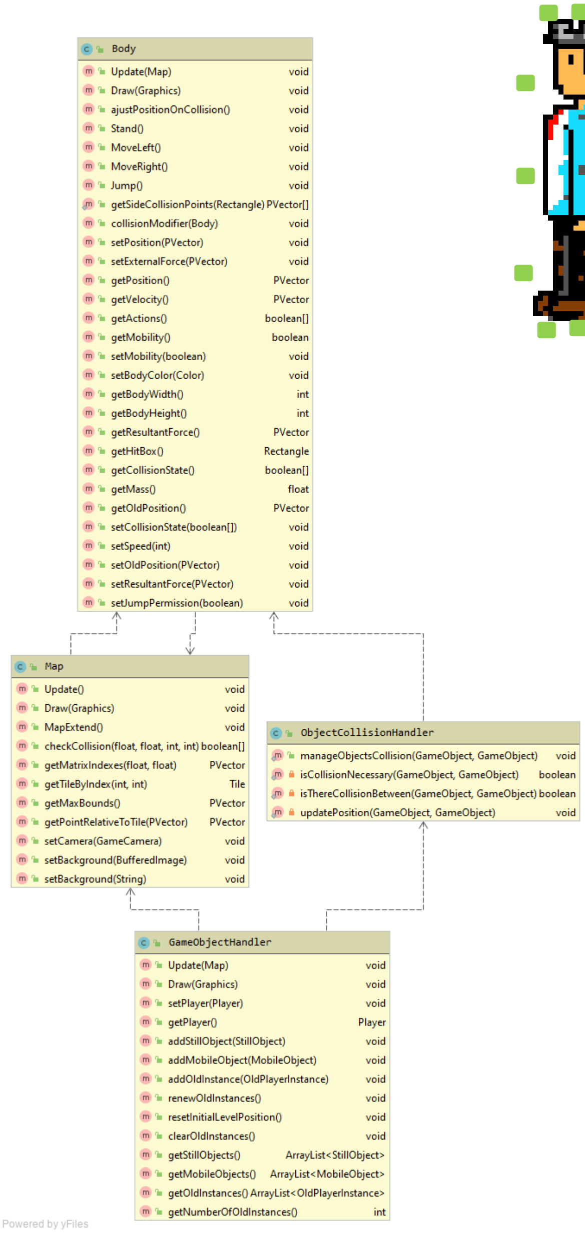
iv) **The collision** is handled from two different perspectives: between objects and map and between objects.

A required information that was needed for implementing the movement was the side on which the collision was taking place. In order to get the side, both types of collisions handle the problem in the same way. On each side of the collision rectangle were considered a number of points where are conducted calculations over the posibility of collision.

For the object – object interraction from the point of collision, a special class called ObjectCollisionHandler was designed. The class has a static method called manageObjectCollision that takes two GameObjects and check for their collision.



These are the points in which the collision is checked for the Player or any other objects. Of course, the number of such points is higher than 10 on each side.



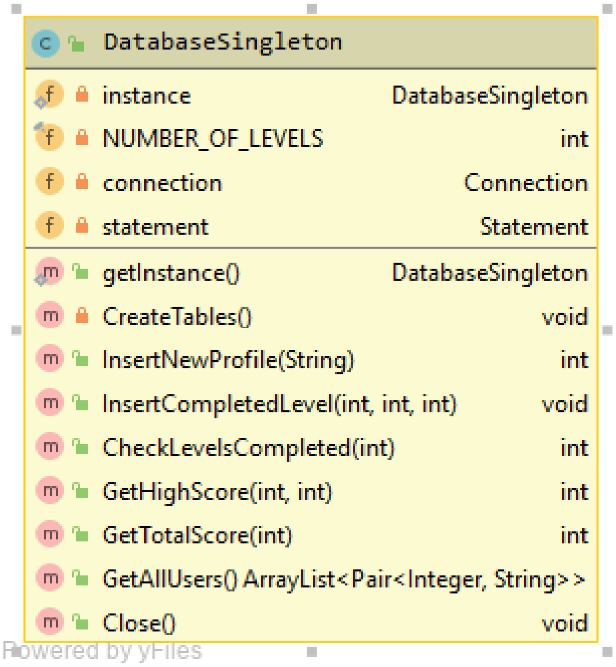
Body	
Update(Map)	void
Draw(Graphics)	void
ajustPositionOnCollision()	void
Stand()	void
MoveLeft()	void
MoveRight()	void
Jump()	void
getSideCollisionPoints(Rectangle)	PVector[]
collisionModifier(Body)	void
setPosition(PVector)	void
setExternalForce(PVector)	void
getPosition()	PVector
getVelocity()	PVector
getActions()	boolean[]
getMobility()	boolean
setMobility(boolean)	void
setBodyColor(Color)	void
getBodyWidth()	int
getBodyHeight()	int
getResultantForce()	PVector
getHitBox()	Rectangle
getCollisionState()	boolean[]
getMass()	float
getOldPosition()	PVector
setCollisionState(boolean[])	void
setSpeed(int)	void
setOldPosition(PVector)	void
setResultantForce(PVector)	void
setJumpPermission(boolean)	void

Map	
Update()	void
Draw(Graphics)	void
MapExtend()	void
checkCollision(float, float, int, int)	boolean[]
getMatrixIndexes(float, float)	PVector
getTileByIndex(int, int)	Tile
getMaxBounds()	PVector
getPointRelativeToTile(PVector)	PVector
setCamera(GameCamera)	void
setBackground(BufferedImage)	void
setBackground(String)	void

ObjectCollisionHandler	
manageObjectsCollision(GameObject, GameObject)	void
isCollisionNecessary(GameObject, GameObject)	boolean
isThereCollisionBetween(GameObject, GameObject)	boolean
updatePosition(GameObject, GameObject)	void

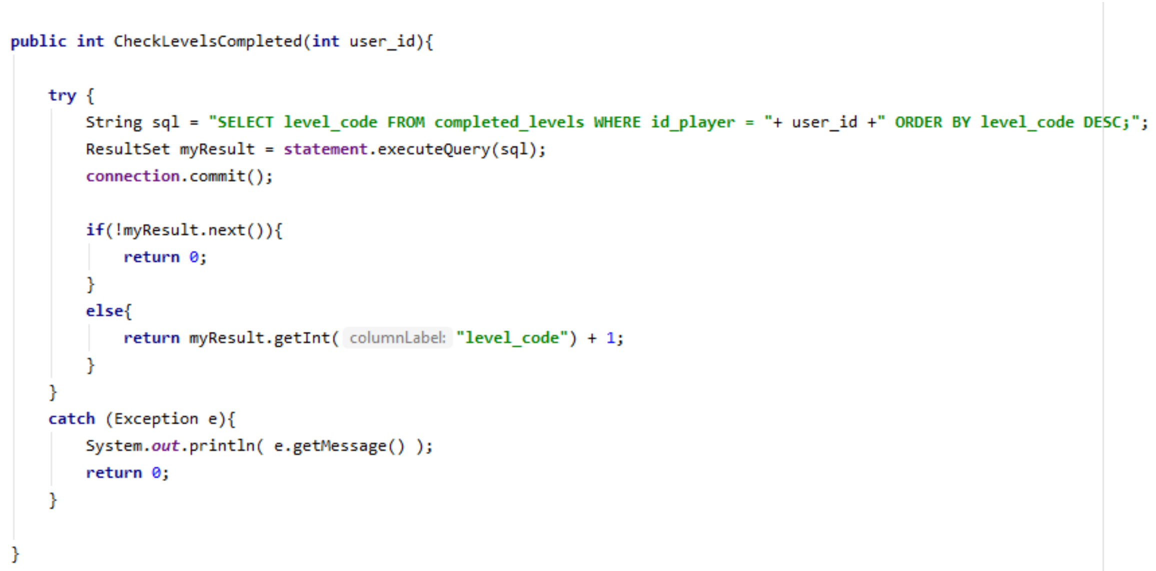
GameObjectHandler	
Update(Map)	void
Draw(Graphics)	void
setPlayer(Player)	void
getPlayer()	Player
addStillObject(StillObject)	void
addMobileObject(MobileObject)	void
addOldInstance(OldPlayerInstance)	void
renewOldInstances()	void
resetInitialLevelPosition()	void
clearOldInstances()	void
getStillObjects()	ArrayList<StillObject>
getMobileObjects()	ArrayList<MobileObject>
getOldInstances()	ArrayList<OldPlayerInstance>
getNumberOfOldInstances()	int

v) **The Database** is managed through a class that is a singleton. Each different querry to the database was incapsulated by different methods of the database class. The saving system of the game is managed through queries to the database.



DatabaseSingleton	
instance	DatabaseSingleton
NUMBER_OF_LEVELS	int
connection	Connection
statement	Statement
getInstance()	DatabaseSingleton
CreateTables()	void
InsertNewProfile(String)	int
InsertCompletedLevel(int, int, int)	void
CheckLevelsCompleted(int)	int
GetHighScore(int, int)	int
GetTotalScore(int)	int
GetAllUsers() ArrayList<Pair<Integer, String>>	
Close()	void

Exception handeling: The project is handelling well the exceptions that may occour during different operations such as: Database queries or IO Exceptions.



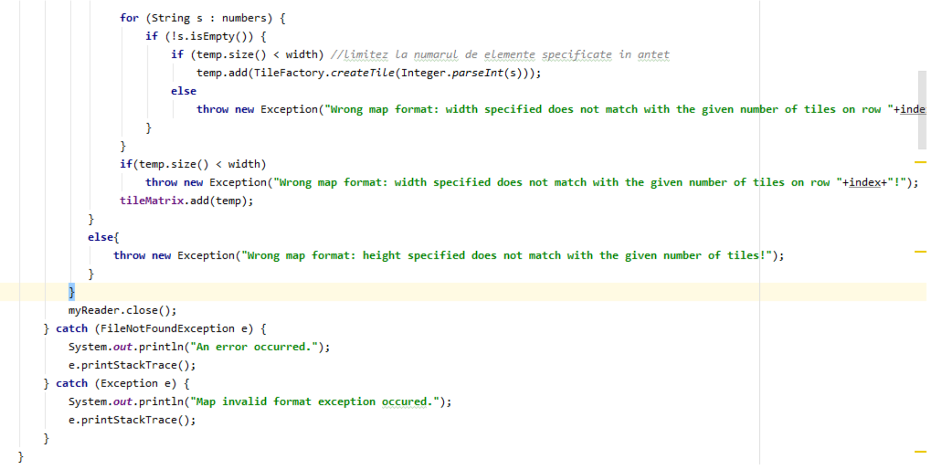
```
public int CheckLevelsCompleted(int user_id){

    try {
        String sql = "SELECT level_code FROM completed_levels WHERE id_player = "+ user_id +" ORDER BY level_code DESC;";
        ResultSet myResult = statement.executeQuery(sql);
        connection.commit();

        if(!myResult.next()){
            return 0;
        }
        else{
            return myResult.getInt( columnLabel: "level_code") + 1;
        }
    }
    catch (Exception e){
        System.out.println( e.getMessage() );
        return 0;
    }

}
```

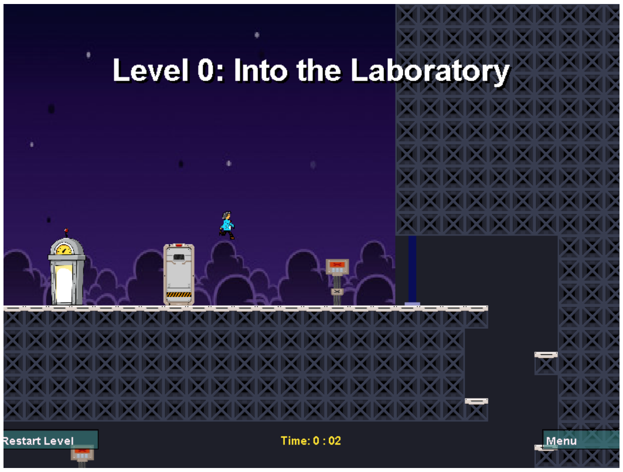
(example from the DatabaseSingleton class)

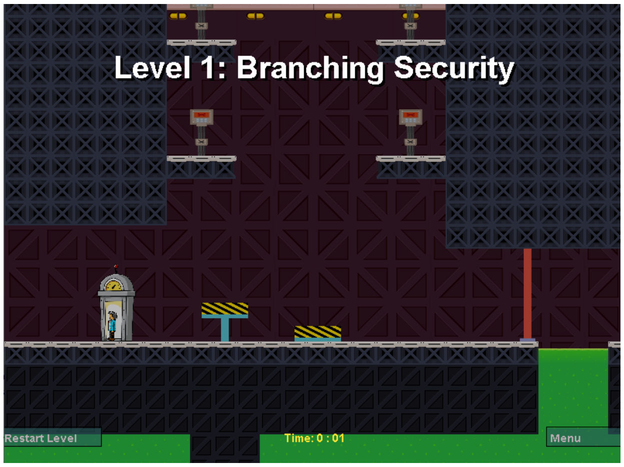


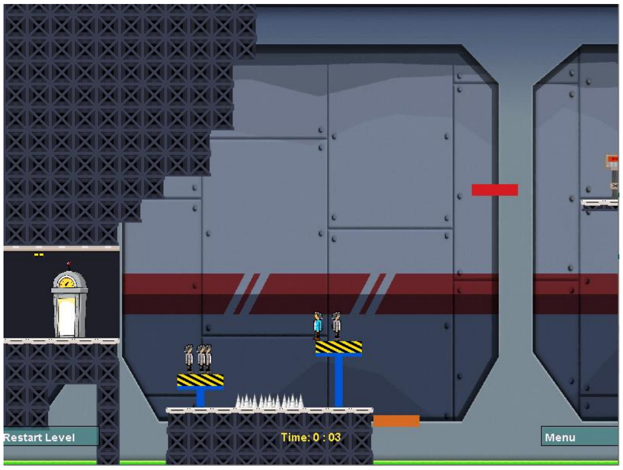
```
                for (String s : numbers) {
                    if (!s.isEmpty()) {
                        if (temp.size() < width) //limitez la numarul de elemente specificate in antet
                            temp.add(TileFactory.createTile(Integer.parseInt(s)));
                        else
                            throw new Exception("Wrong map format: width specified does not match with the given number of tiles on row "+inde
                    }
                }
                if(temp.size() < width)
                    throw new Exception("Wrong map format: width specified does not match with the given number of tiles on row "+index+"!");
                tileMatrix.add(temp);
            }
            else{
                throw new Exception("Wrong map format: height specified does not match with the given number of tiles!");
            }
        }
        myReader.close();
    } catch (FileNotFoundException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    } catch (Exception e) {
        System.out.println("Map invalid format exception occured.");
        e.printStackTrace();
    }
}
```

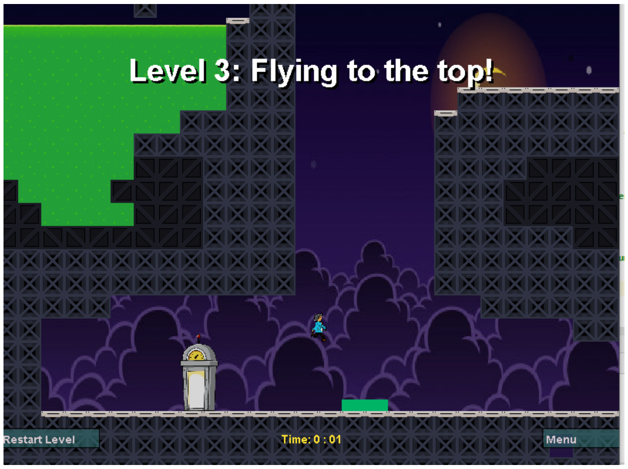
(example from the Map class where it is read a file for the tiles)

In game screenshots:









Gamplay link on Youtube: https://youtu.be/KVL3RTZMS3U