Term Project Documentation

CSC 413-03

Spring 2019

Victor Villalpando

Table of Contents

1. Introduction

1.1 Project Overview

In this term project report, I will discuss the similarities and differences between the implementation of Tank Wars and Pyramid Panic games. For the second game, I decided to implement the Pyramid Panic game because I wanted to challenge my programming capabilities since the tank game was not difficult.

1.2 Tank Wars Introduction

Tank Wars is a two-player tank game, in which two players engage in tank battle and attempt to attack each other until one the players remain standing. From the start, each tank has three lives and five health points. Also, each tank has unlimited bullets that deplete one health point on the rival tank. To make the game more interesting breakable and unbreakable walls and health power ups are strategically placed across the game map. Players can destroy breakable walls with two bullet impacts and pick up life power ups if they do not exceed three lives. When a player kills the rival tank the game ends. If the players wish to play again, they must close the game and run it again.

1.3 Pyramid Panic Introduction

In Pyramid Panic, the player is an explorer who has become trapped while investigating a large pyramid. The main objective of the game is to collect as much as treasure as possible and escape the pyramid alive after collecting the biggest treasure, the Sword of Ra. Collecting the treasure of ancient pharaoh will not be easy, beetles and scorpions will block the player's path and mummies will follow the player's steps. Beetles move up and down at a constant speed, but if the player gets too close, they will double their speed. Scorpions behave just like beetles, but they move from left to right. Mummies do not move in any direction, unless they detect the player; they will try to hunt the player down by following its every step.

To help the player in the treasure hunt scarabs, potion and pistol power ups are strategically placed across the map. The player can pick up scarabs to use them to scare the mummies and make them walk away for 10 seconds. While the mummies are scared the player can hunt them down. Potion and pistol power ups grant the player an extra life and a pistol with three bullets, respectively. The player can use the pistol to kill beetles, scorpions, and mummies only when they are scared. In addition to power ups, the player can use horizontal and vertical movable blocks to hide from enemies or access other areas. However, if the blocks are moved too far from their original position and touch a wall or another movable block the block(s) will return their original position.

Deep in the pyramid, there lies the Sword of Ra. When the player picks up the sword the pyramid will plunged into darkness, but the small glow of the sword will light the player's way. The player must scape from the pyramid alive, however, going back to the pyramid entrance can be challenging. The player can temporarily activate the light that sword emits to light the it's way and scare mummies, but at the cost of losing score points.

2. Development Environment

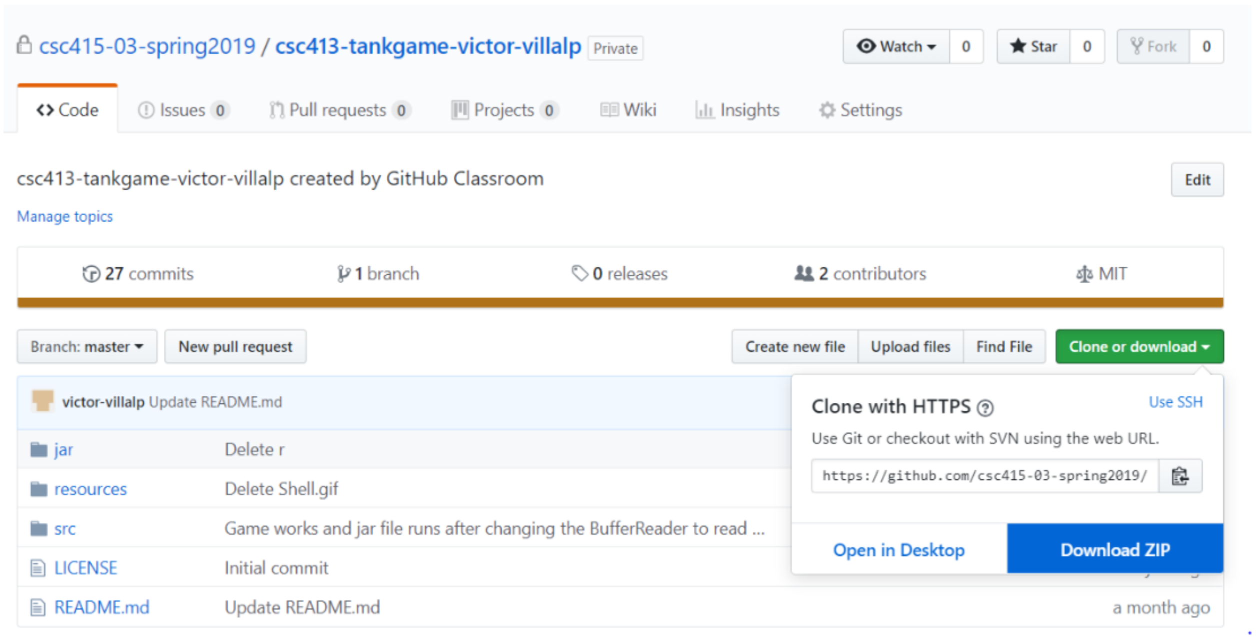
a. Java Version 11.0.1 was used.
b. IDE used: IntelliJ IDEA.
c. No special libraries were used in the development of either game.

3. How to Import the Games from GitHub

3.1 Import and Run Games

The instructions listed below demonstrate how to import a game project to IntelliJ IDEA and run it, using the tank game as an example. Repeat the same steps to import and run the second game, the Pyramid Panic.

1. Click the **Clone or download** button in the game repository link as shown below.


csc415-03-spring2019 / csc413-tankgame-victor-villalp Private

Watch 0 Star 0 Fork 0

Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Insights | Settings

csc413-tankgame-victor-villalp created by GitHub Classroom

Edit

Manage topics

27 commits | 1 branch | 0 releases | 2 contributors | MIT

Branch: master | New pull request | Create new file | Upload files | Find File | Clone or download

victor-villalp Update README.md

jar	Delete r
resources	Delete Shell.gif
src	Game works and jar file runs after changing the BufferReader to read ...
LICENSE	Initial commit
README.md	Update README.md

a month ago

Clone with HTTPS | Use SSH

Use Git or checkout with SVN using the web URL.

```
https://github.com/csc415-03-spring2019/
```

Open in Desktop | Download ZIP

Figure 3.1-1. GitHub Tank Game Repository Page

2. After downloading the game from the repo link, open IntelliJ IDEA and click the **Import Project** text field.

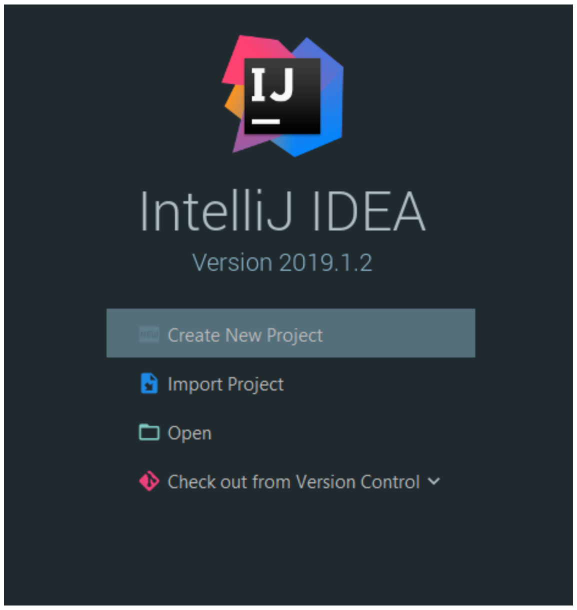
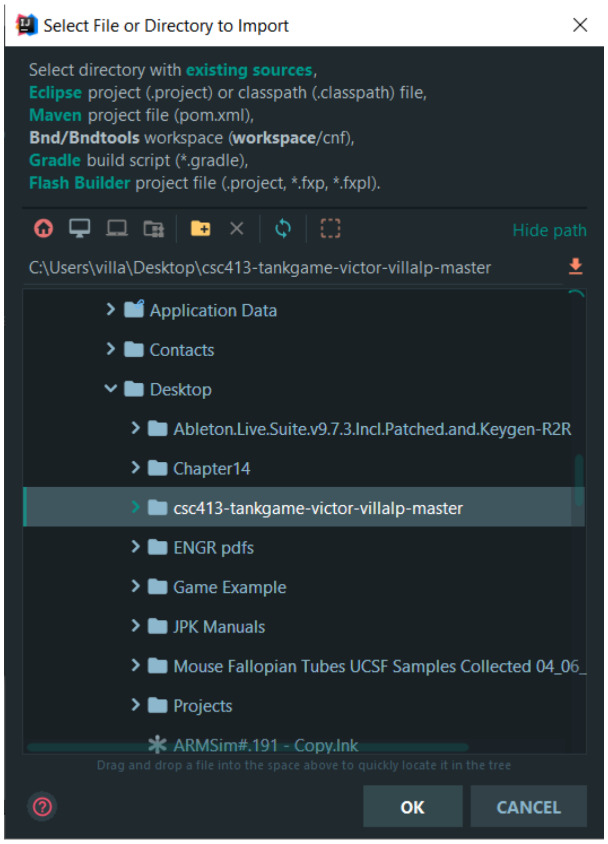


Figure 3.1-2. IntelliJ IDEA Project Window

3. Click the **Import Project** text field to open the **Select File or Directory to Import** window. Select or paste the full path where the game was downloaded to in the text field shown below and click **Ok** to proceed


Select File or Directory to Import

Select directory with existing sources,
Eclipse project (.project) or classpath (.classpath) file,
Maven project file (pom.xml),
Bnd/Bndtools workspace (workspace/cnf),
Gradle build script (*.gradle),
Flash Builder project file (.project, *.fxp, *.fxpl).

C:\Users\villa\Desktop\csc413-tankgame-victor-villalp-master

Hide path
Application Data
Contacts
Desktop
Ableton.Live.Suite.v9.7.3.Incl.Patched.and.Keygen-R2R
Chapter14
csc413-tankgame-victor-villalp-master
ENGR pdfs
Game Example
JPK Manuals
Mouse Fallopian Tubes UCSF Samples Collected 04_06_
Projects
ARMSim#.191 - Copy.lnk
Drag and drop a file into the space above to quickly locate it in the tree
OK
CANCEL

Figure 3.1-3. Select File or Directory to Import Window

4. Select **Create project from existing sources** and click **Next** to continue.

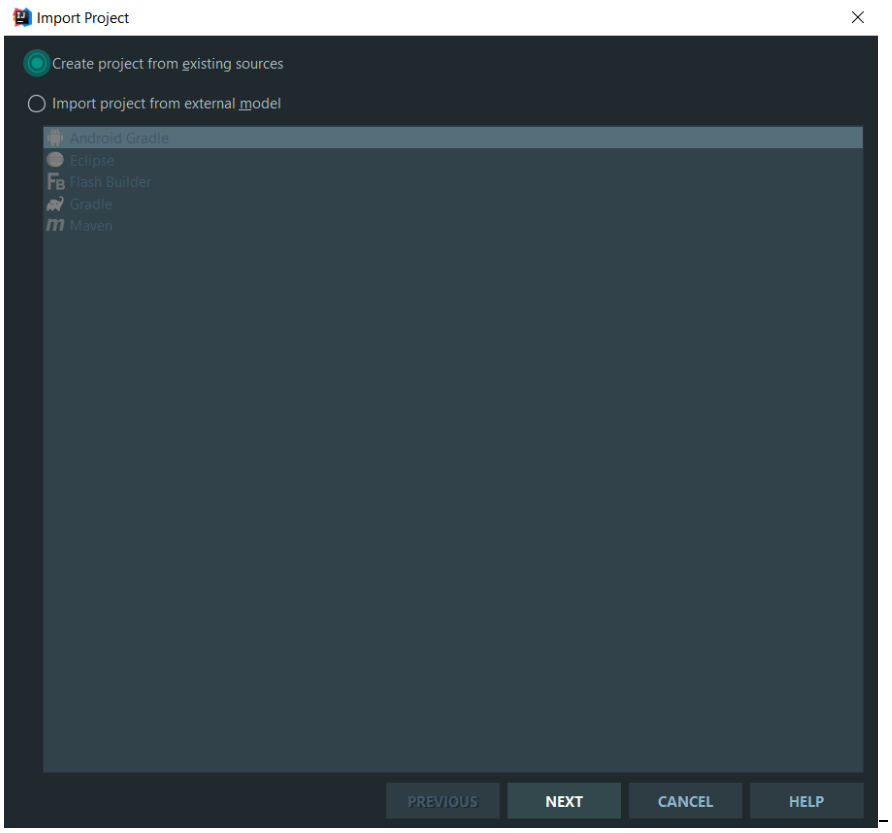
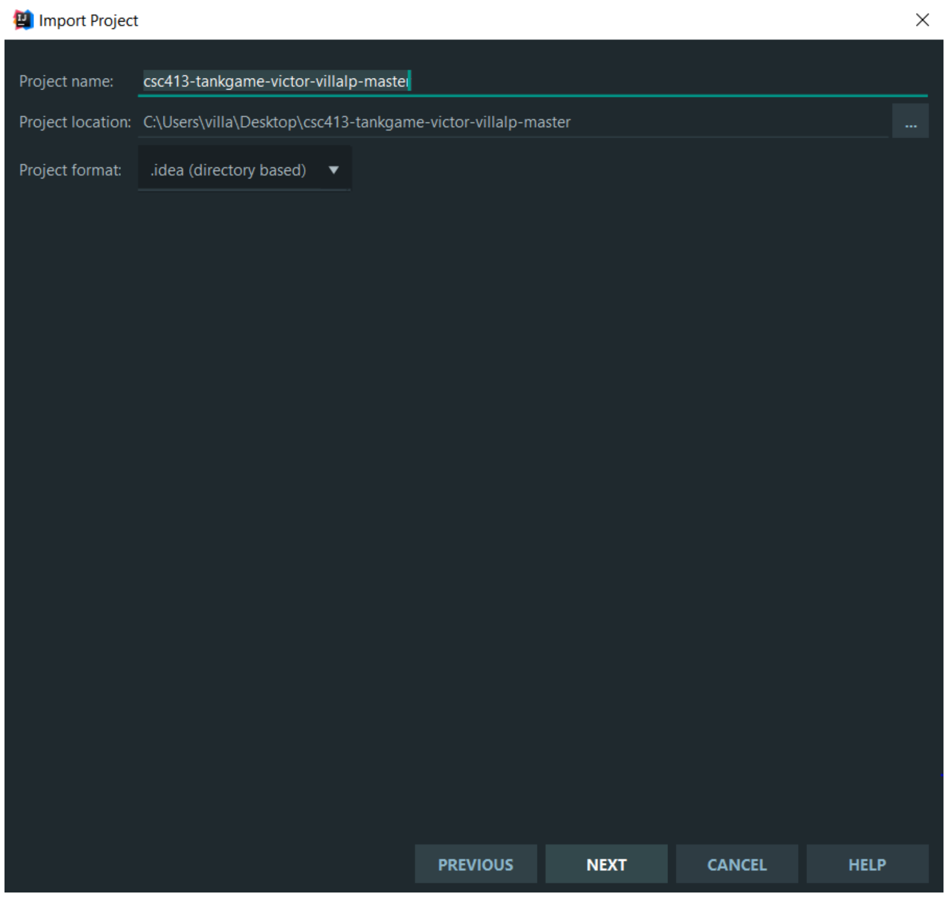


Figure 3.1-4. Import Project Window

5. Adding the path of the downloaded project will keep the project's original name, but the project can be renamed if you wish, otherwise click **Next** to continue.

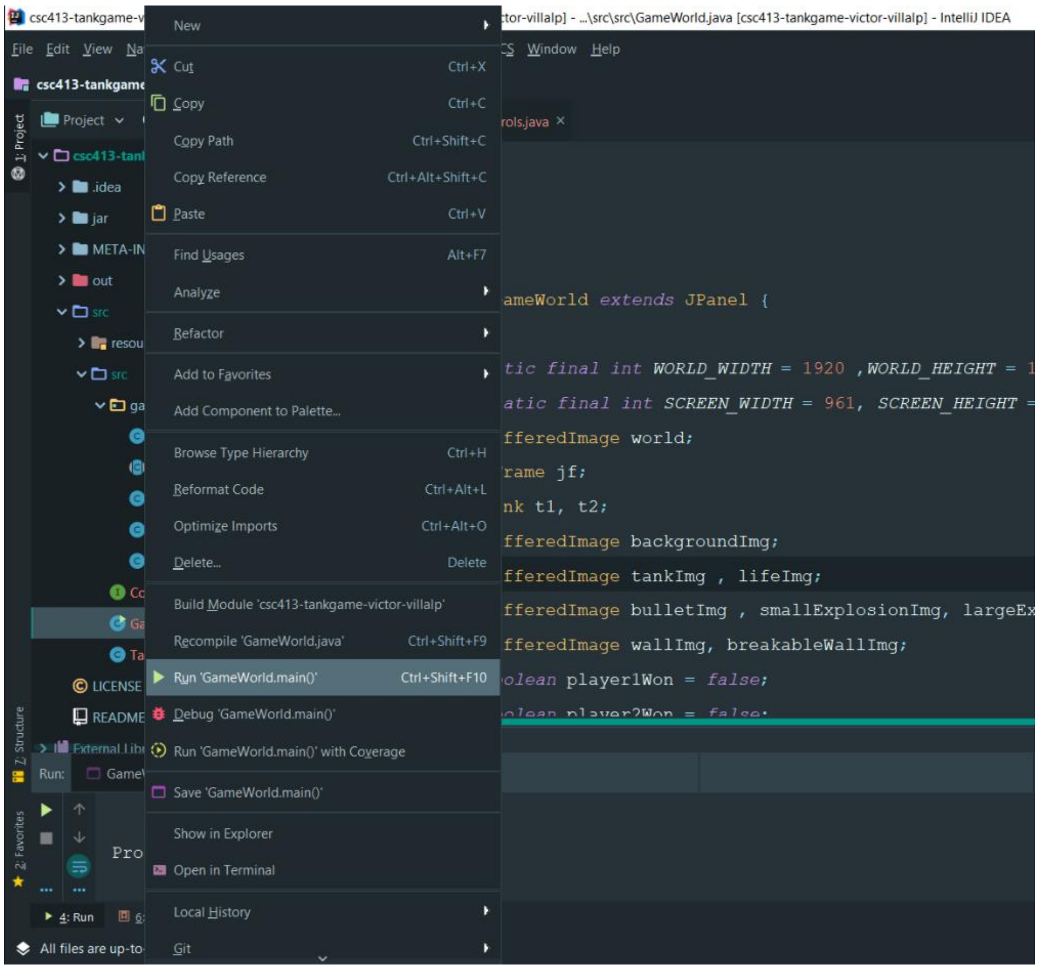


Project name: csc413-tankgame-victor-villalp-master
Project location: C:\Users\villa\Desktop\csc413-tankgame-victor-villalp-master
Project format: .idea (directory based)

Figure 3.1-5. Project Name Window

6. Click **Next** for all windows until reaching the **Please select project SDK**. Select a java version for the project and click **Next.** Lastly click **Finish** to create the project.

7. After successfully creating an IntelliJ project from my existing game project, run the tank game by right clicking on the "GameWorld" main class and then clicking on the "Run GameWorld.main()" box as shown below.

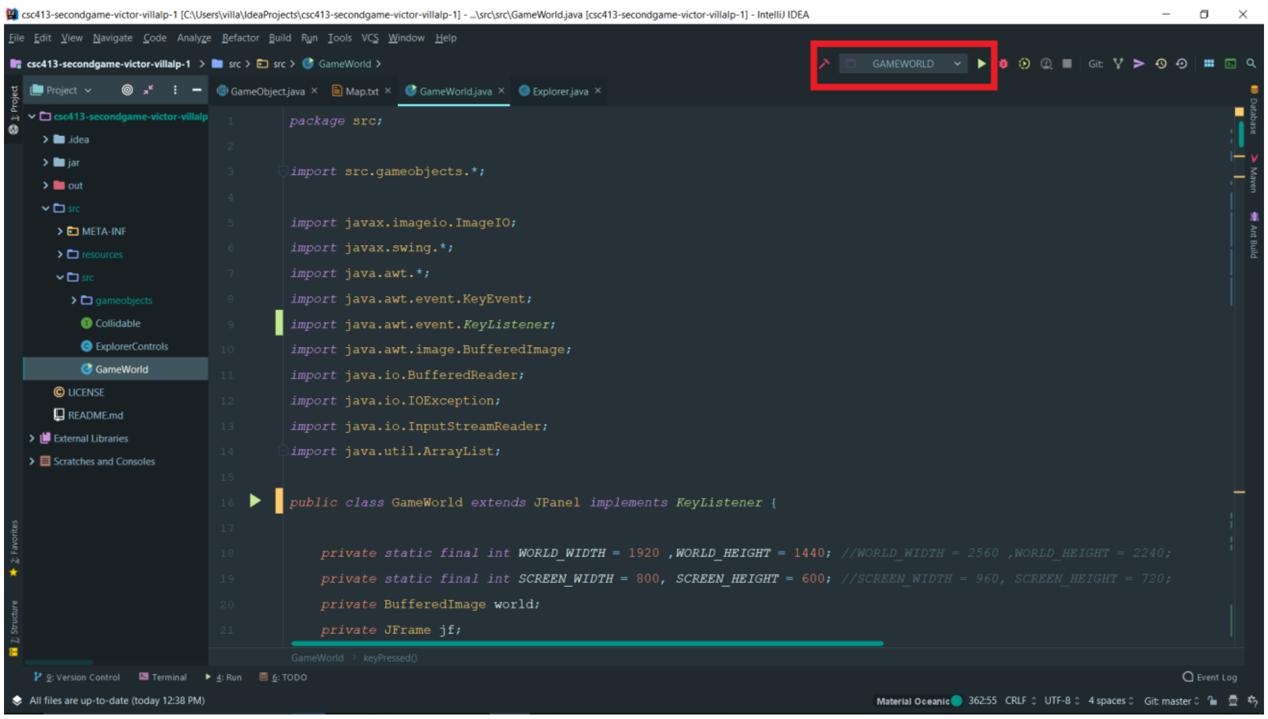


```
ameWorld extends JPanel {

tic final int WORLD_WIDTH = 1920 ,WORLD_HEIGHT = 1
atic final int SCREEN_WIDTH = 961, SCREEN_HEIGHT =
fferedImage world;
rame jf;
nk t1, t2;
fferedImage backgroundImg;
fferedImage tankImg , lifeImg;
fferedImage bulletImg , smallExplosionImg, largeEx
fferedImage wallImg, breakableWallImg;
olean player1Won = false;
olean player2Won = false;
```

```
Pro
```

Figure 3.1-6. Run “GameWorld.main()”

8. The game can also be run by simply clicking the green play button located in the upper right corner of the IntelliJ IDEA window as shown in Figure 3.1-7.



```
package src;

import src.gameobjects.*;

import javax.imageio.ImageIO;
import javax.swing.*;
import java.awt.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.image.BufferedImage;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;

public class GameWorld extends JPanel implements KeyListener {

    private static final int WORLD_WIDTH = 1920 ,WORLD_HEIGHT = 1440; //WORLD_WIDTH = 2560 ,WORLD_HEIGHT = 2240;
    private static final int SCREEN_WIDTH = 800, SCREEN_HEIGHT = 600; //SCREEN_WIDTH = 960, SCREEN_HEIGHT = 720;
    private BufferedImage world;
    private JFrame jf;
```

Figure 3.1-7. Green Play Button Icon

9. When the game is successfully built the window shown in Figure 3.1-9 will appear in your computer screen and you will be able to engage with the game controls as described in the next section.

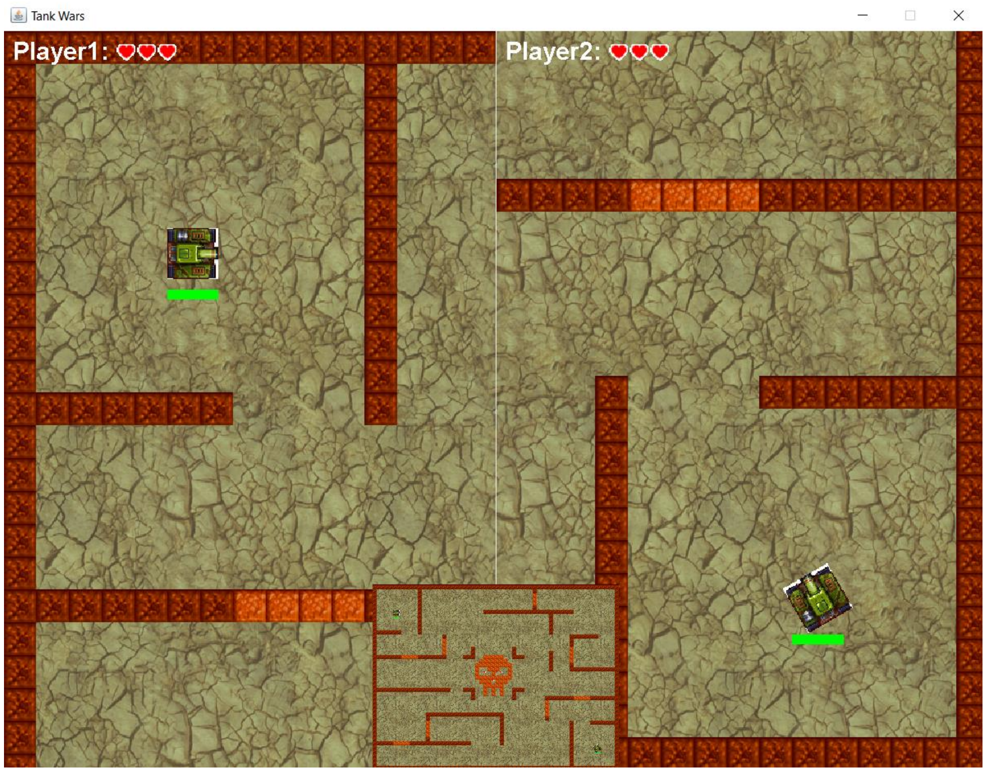


Figure 3.1-9. “Tank Wars” Game Window

10. Repeat steps 1-9 to import and run the second game, Pyramid Panic. If done correctly the game window shown in Figure 3.1-10 will appear in your screen.

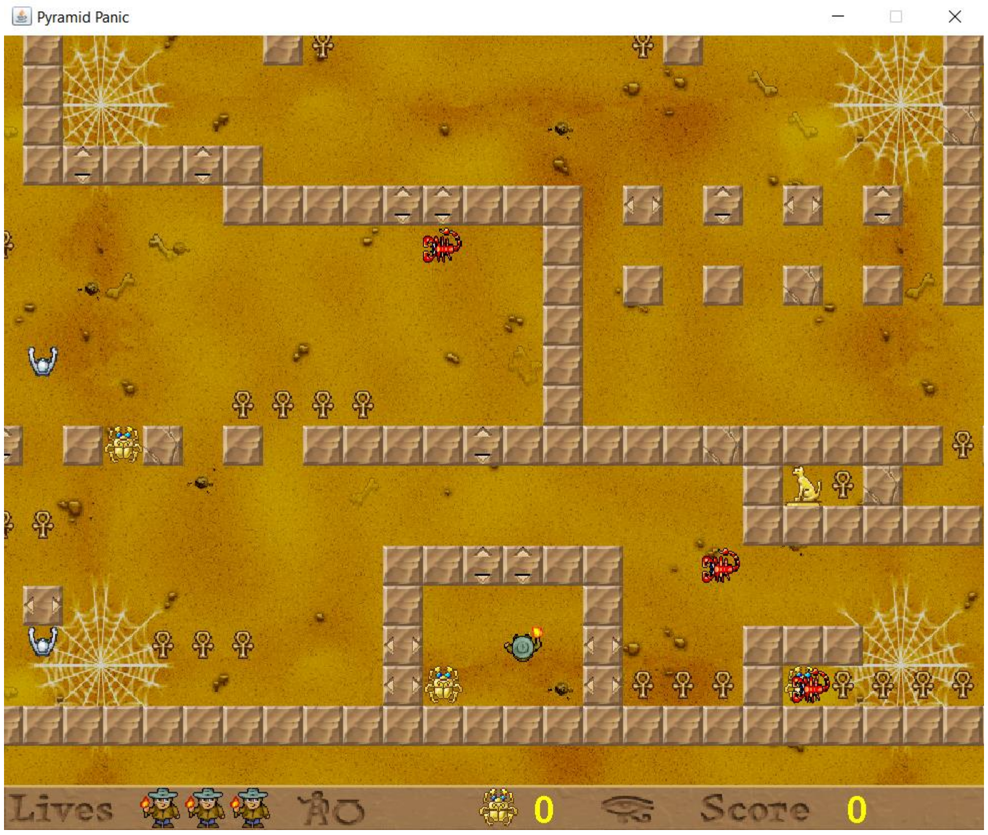
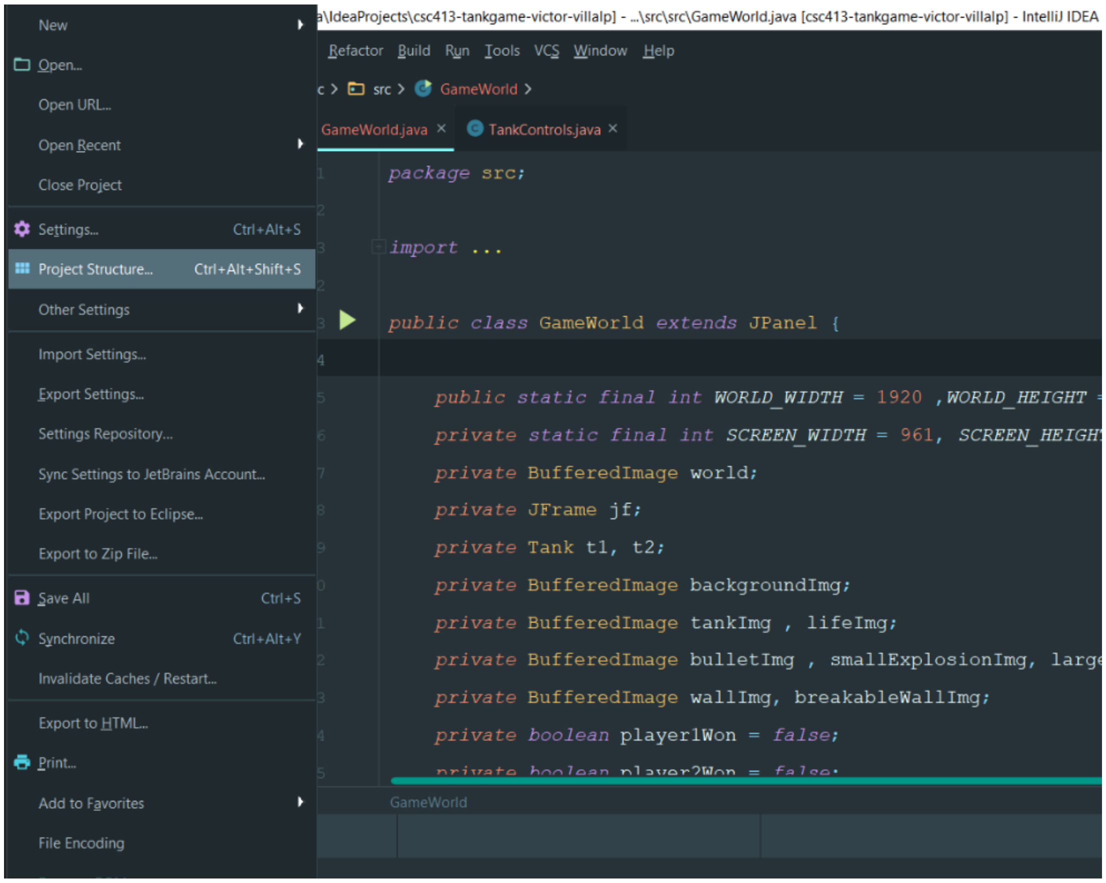


Figure 3.1-10. “Pyramid Panic” Game Window

3.2 Build Game JAR

After successfully importing and running both games, now it is time to build a JAR file for each game project. Follow the steps below to first build a jar file for the tank game and repeat the same instructions for Pyramid Panic.

1. Click the **File** text filed located in the upper left most corner of the IntelliJ IDEA window to open the drop-down list shown in the figure below. Then click on the “Project Structure” box.



```
package src;

import ...

public class GameWorld extends JPanel {

    public static final int WORLD_WIDTH = 1920 ,WORLD_HEIGHT =
    private static final int SCREEN_WIDTH = 961,  SCREEN_HEIGH
    private BufferedImage world;
    private JFrame jf;
    private Tank t1, t2;
    private BufferedImage backgroundImg;
    private BufferedImage tankImg , lifeImg;
    private BufferedImage bulletImg , smallExplosionImg, large
    private BufferedImage wallImg, breakableWallImg;
    private boolean player1Won = false;
    private boolean player2Won = false;
```

Figure 3.2-1. File Drop-Down List

2. Clicking on the “Project Structure” will open a “Project Structure” window. Click on the “Artifacts” box as shown in the figure below.

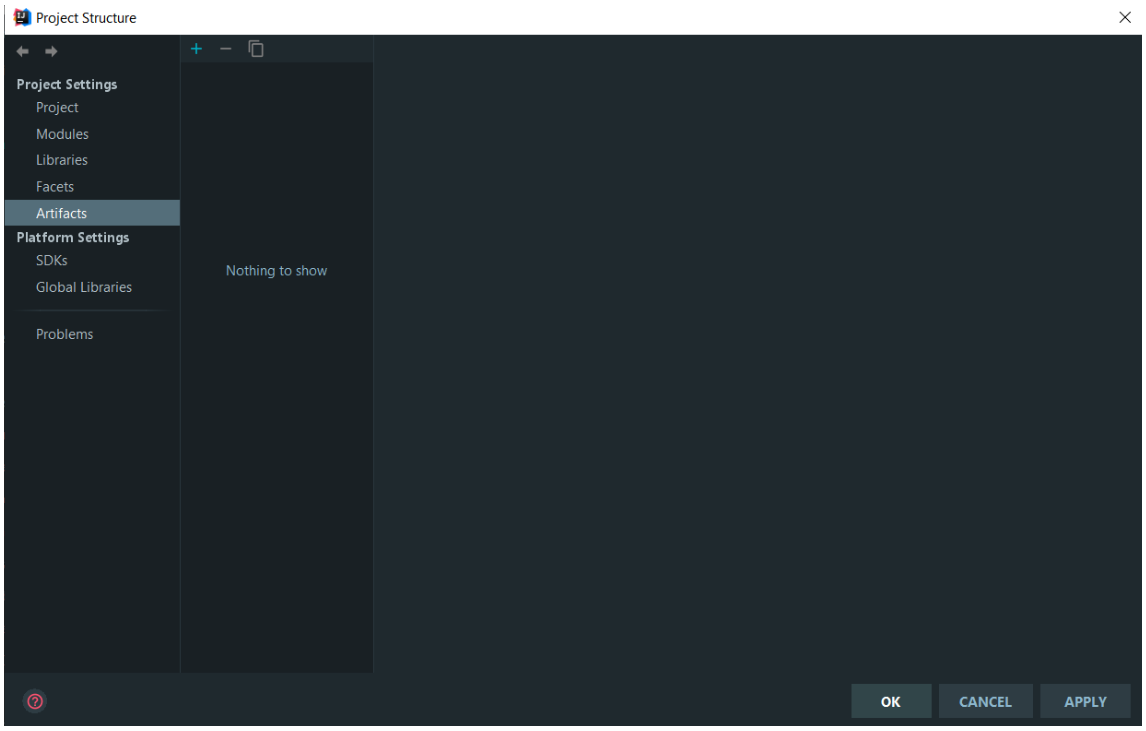


Figure 3.2-2. Project Structure Window

3. Click on the “+” icon located in the upper left corner of the next shaded section to open the list of artifacts that can be added to the project. Then, select “JAR” and “From modules with dependencies” as shown below.

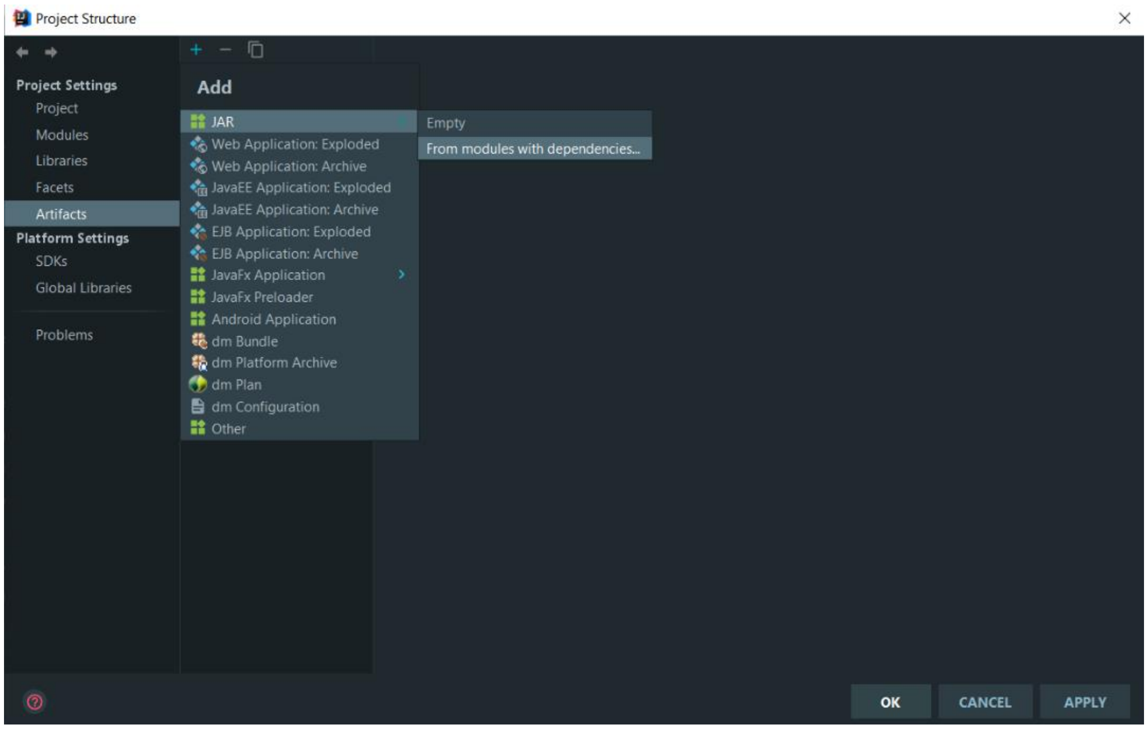
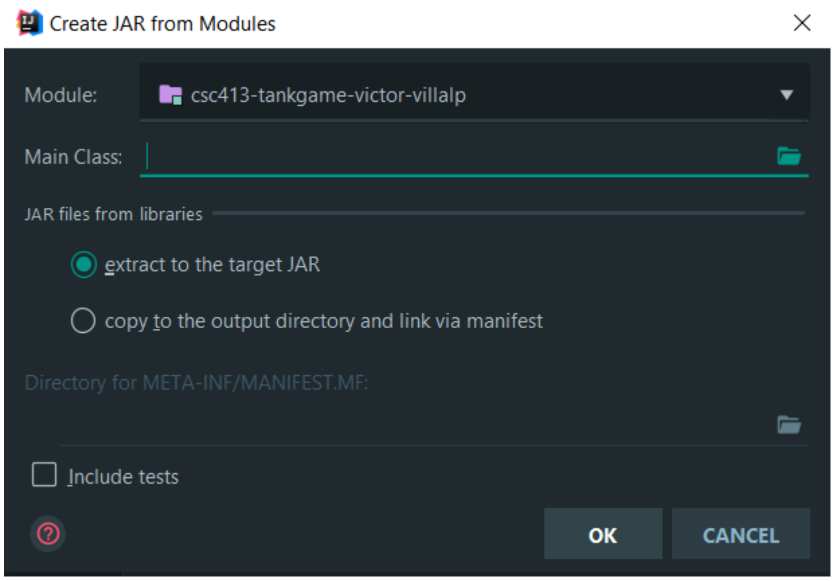


Figure 3.2-3 JAR Artifact “From modules with dependencies”

4. After selecting "JAR" and "from modules with dependencies," a "Create JAR from Modules" window will pop up. Then, click on the folder icon and select GameWorld as the main class as shown in Figure 3.2-5.



Create JAR from Modules

Module: csc413-tankgame-victor-villalp

Main Class:

JAR files from libraries

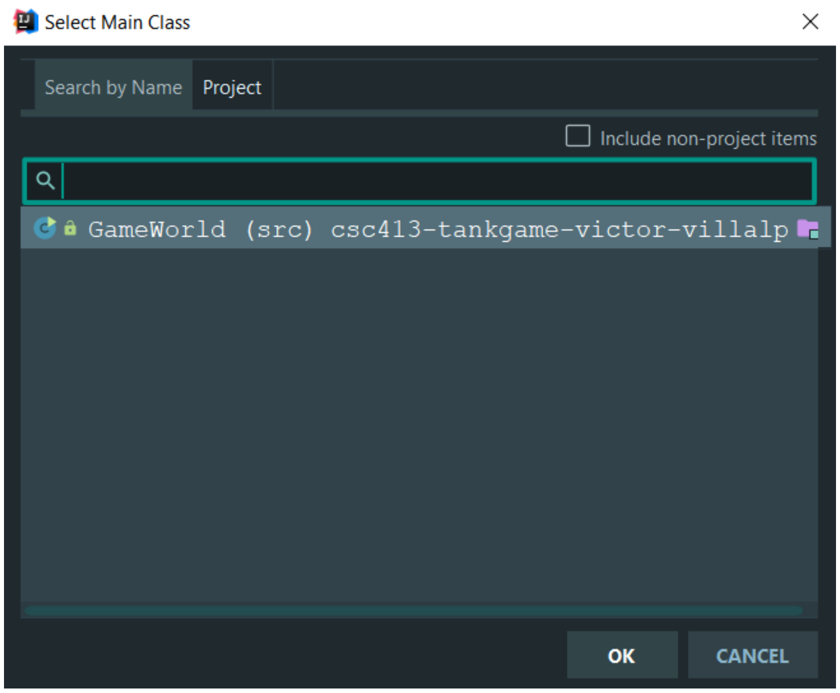
(●) extract to the target JAR

( ) copy to the output directory and link via manifest

Directory for META-INF/MANIFEST.MF:

☐ Include tests

Figure 3.2-4. Create JAR from Modules Window



Select Main Class

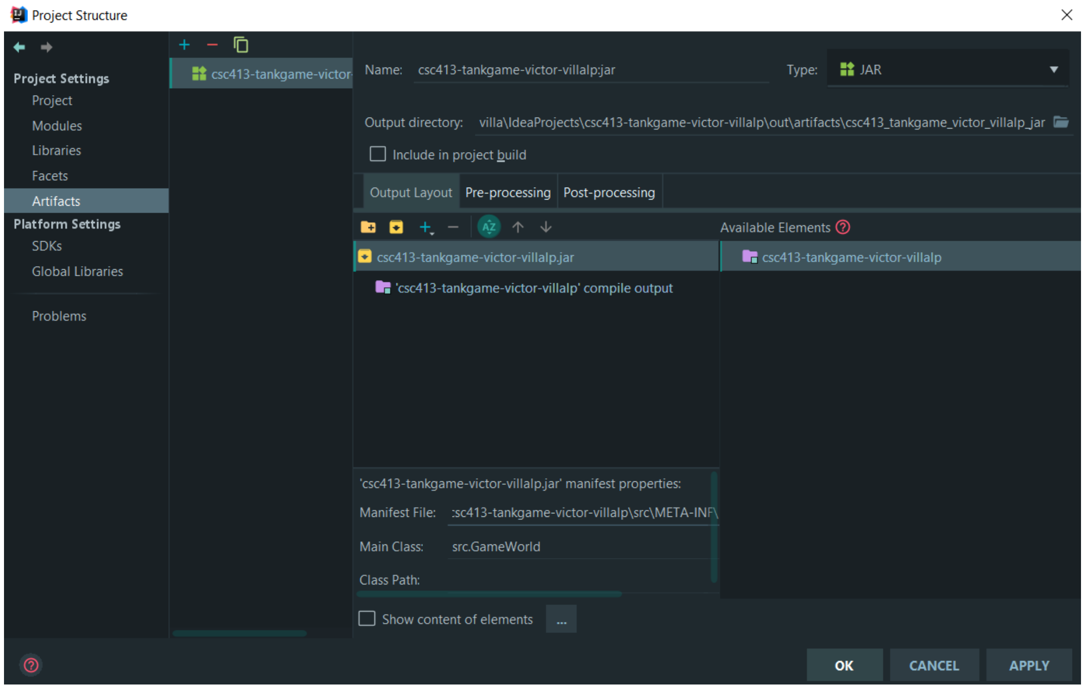
☐ Include non-project items

```
GameWorld (src) csc413-tankgame-victor-villalp
```

Figure 3.2-5 Select Main Class

5. When GameWorld is selected as the main class the "Select Main Class" window will close. Click **OK** at the right bottom corner of the "Create JAR from Modules" window

add the artifact. The “Project Structure” will resemble the figure below when the JAR artifact is added to the project.


Project Structure

Name: csc413-tankgame-victor-villalp:jar
Type: JAR
Output directory: villa\IdeaProjects\csc413-tankgame-victor-villalp\out\artifacts\csc413_tankgame_victor_villalp_jar
Include in project build: unchecked

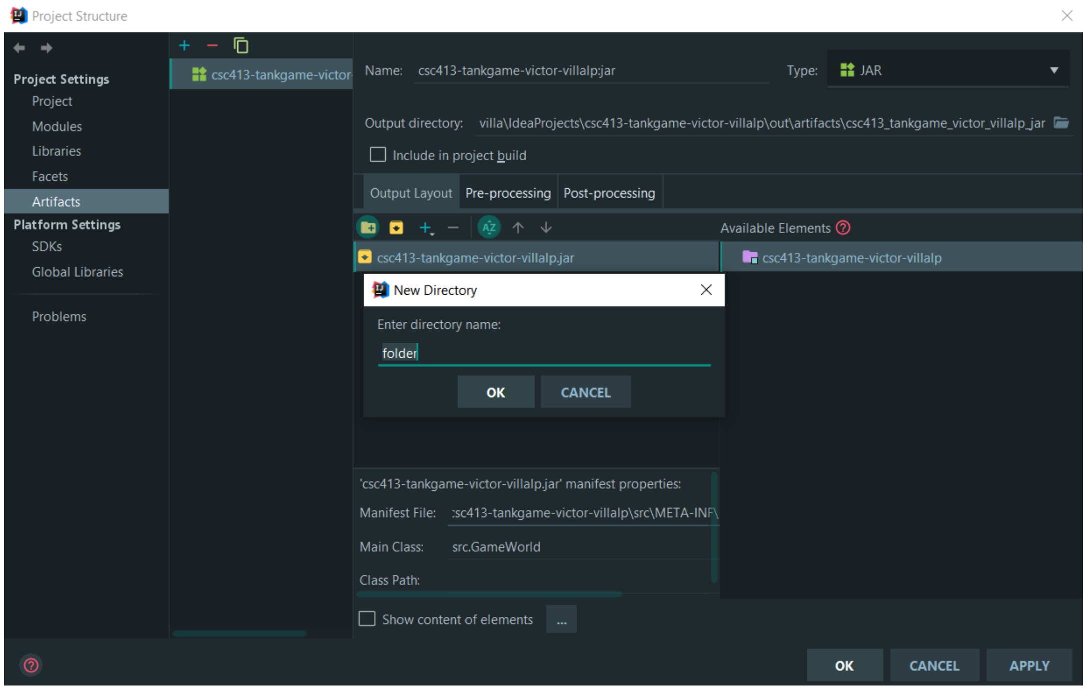
Output Layout:
- csc413-tankgame-victor-villalp.jar
  - 'csc413-tankgame-victor-villalp' compile output

Available Elements:
- csc413-tankgame-victor-villalp

'csc413-tankgame-victor-villalp.jar' manifest properties:
Manifest File: :sc413-tankgame-victor-villalp\src\META-INF\
Main Class: src.GameWorld
Class Path:

Figure 3.2-6 Project Stricture Window with JAR Artifact

6. If you wish to change the output location of the JAR file, click on the first folder icon under the “Output Layout” section and type the name of the desired path as shown in Figure 3.2-7. Click **OK** once again to close the “Project Structure” window.


Project Structure

Name: csc413-tankgame-victor-villalp:jar
Type: JAR
Output directory: villa\IdeaProjects\csc413-tankgame-victor-villalp\out\artifacts\csc413_tankgame_victor_villalp_jar
Include in project build: unchecked

Output Layout:
- csc413-tankgame-victor-villalp.jar

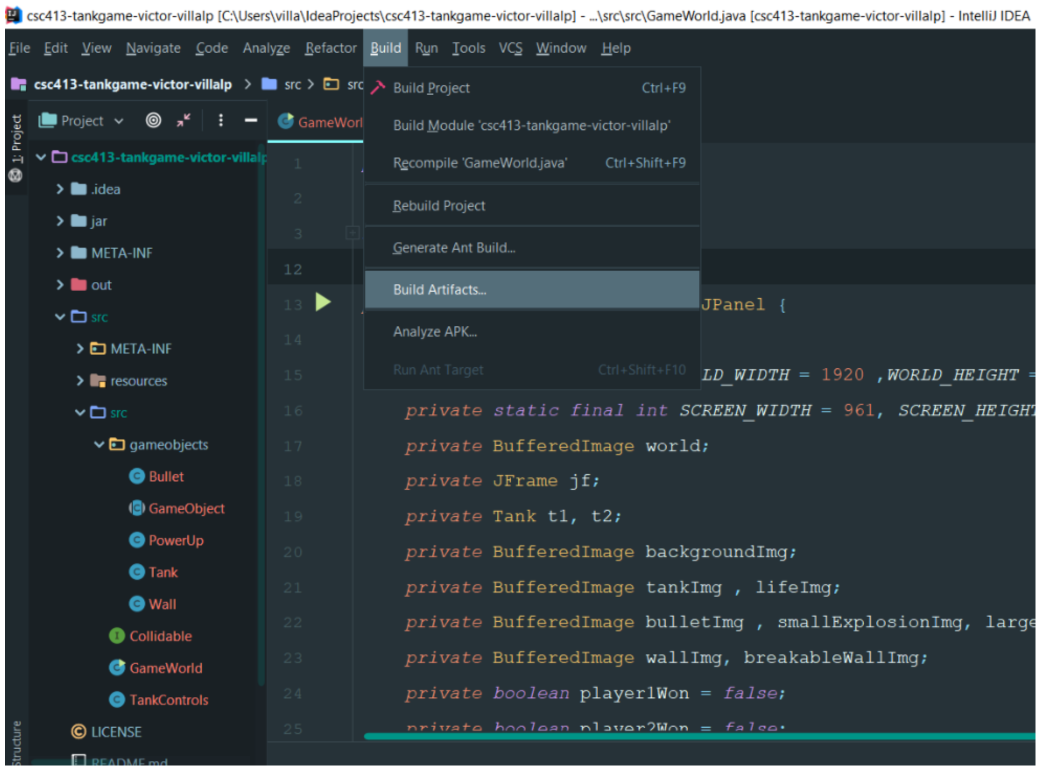
Available Elements:
- csc413-tankgame-victor-villalp

New Directory
Enter directory name: folder

'csc413-tankgame-victor-villalp.jar' manifest properties:
Manifest File: :sc413-tankgame-victor-villalp\src\META-INF\
Main Class: src.GameWorld
Class Path:

Figure 3.2-7. New Directory Window

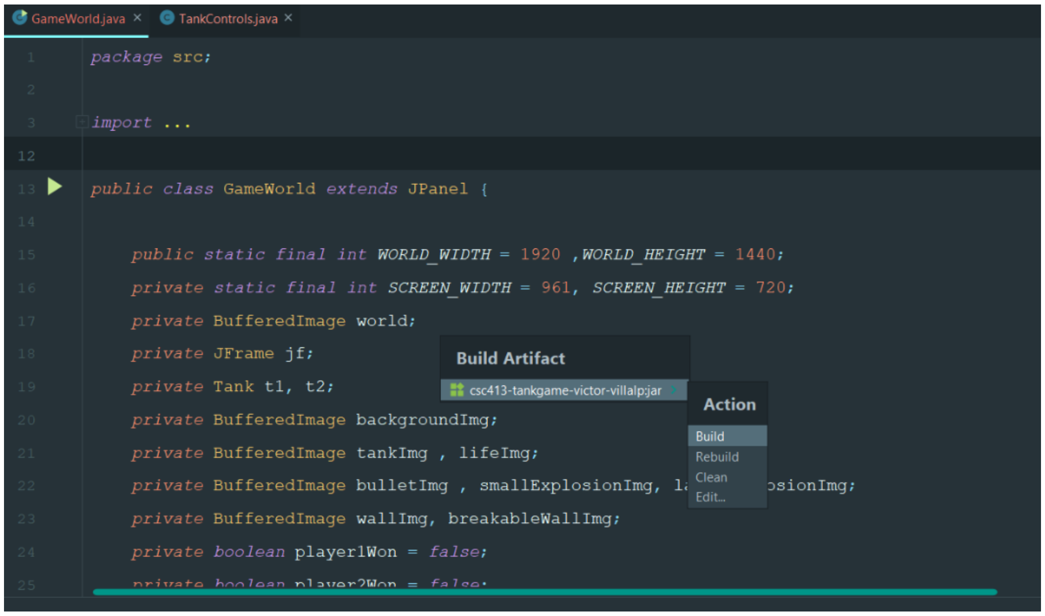
7. Click on the **Build** box located in the upper left section of the IntelliJ IDEA window to open the drop-down list shown in the figure below. Then click on **Build Artifacts**.


csc413-tankgame-victor-villalp [C:\Users\villa\IdeaProjects\csc413-tankgame-victor-villalp] - ...\src\src\GameWorld.java [csc413-tankgame-victor-villalp] - IntelliJ IDEA
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
csc413-tankgame-victor-villalp > src > src
Project
1: Project
csc413-tankgame-victor-villalp
.idea
jar
META-INF
out
src
META-INF
resources
src
gameobjects
Bullet
GameObject
PowerUp
Tank
Wall
Collidable
GameWorld
TankControls
LICENSE
README.md
Structure
GameWorl
Build Project Ctrl+F9
Build Module 'csc413-tankgame-victor-villalp'
Recompile 'GameWorld.java' Ctrl+Shift+F9
Rebuild Project
Generate Ant Build...
Build Artifacts...
Analyze APK...
Run Ant Target Ctrl+Shift+F10

```
JPanel {
LD_WIDTH = 1920 ,WORLD_HEIGHT =
    private static final int SCREEN_WIDTH = 961, SCREEN_HEIGHT
    private BufferedImage world;
    private JFrame jf;
    private Tank t1, t2;
    private BufferedImage backgroundImg;
    private BufferedImage tankImg , lifeImg;
    private BufferedImage bulletImg , smallExplosionImg, large
    private BufferedImage wallImg, breakableWallImg;
    private boolean player1Won = false;
    private boolean player2Won = false;
```

Figure 3.2-8. Build Artifacts

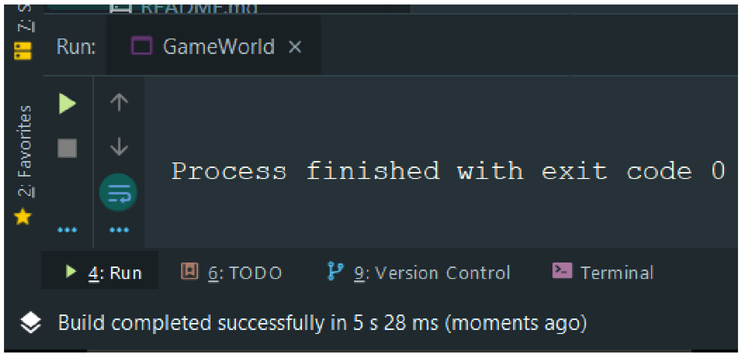
8. Clicking on **Build Artifacts** will open a small “Build Artifact” and “Action” window as shown in Figure 3.2-9. Finally, click on **Build** to build the game jar.



```
package src;

import ...

public class GameWorld extends JPanel {

    public static final int WORLD_WIDTH = 1920 ,WORLD_HEIGHT = 1440;
    private static final int SCREEN_WIDTH = 961, SCREEN_HEIGHT = 720;
    private BufferedImage world;
    private JFrame jf;
    private Tank t1, t2;
    private BufferedImage backgroundImg;
    private BufferedImage tankImg , lifeImg;
    private BufferedImage bulletImg , smallExplosionImg, la      osionImg;
    private BufferedImage wallImg, breakableWallImg;
    private boolean player1Won = false;
    private boolean player2Won = false;
```

Figure 3.2-9. Build Artifact and Action Windows

9. If the previous steps listed in this section were done correctly the JAR artifact will be successfully built and stored in the directory specified in step 6.



```
Process finished with exit code 0
```

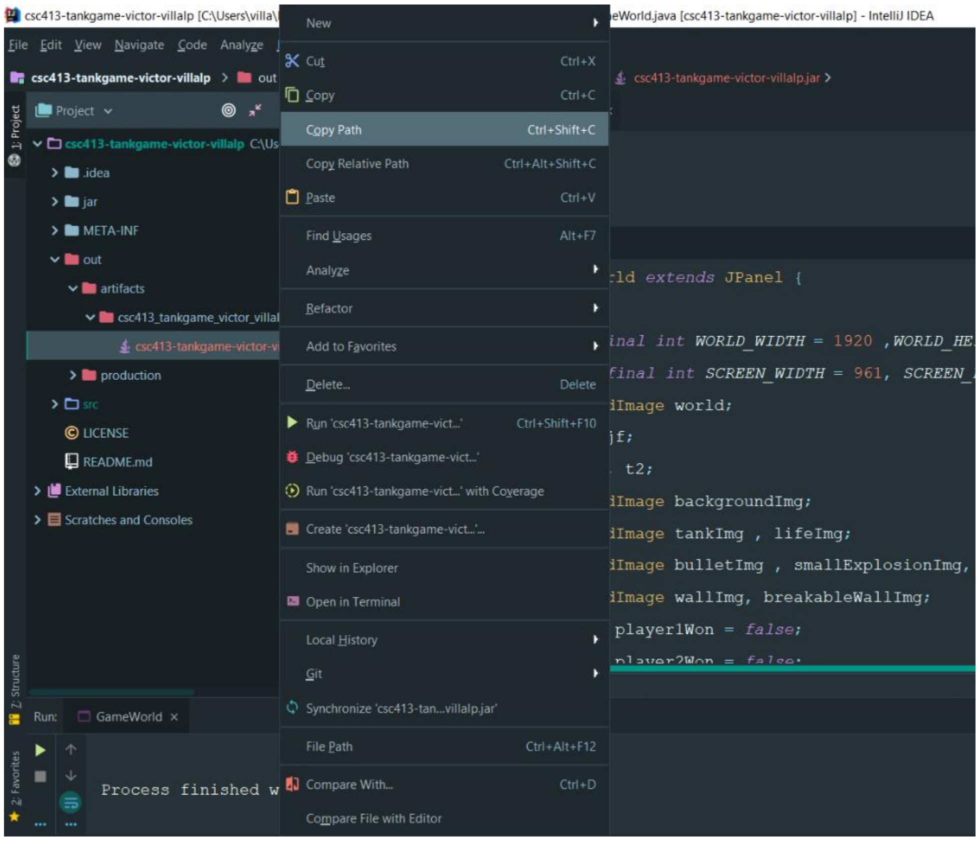
Build completed successfully in 5 s 28 ms (moments ago)

Figure 3.2-10. Successful JAR Build

3.3 Execute JAR

Follow the instructions listed below to execute the tank game JAR file from the Command Prompt; the file will not run by simply clicking on the name file. Repeat the same steps to execute the Pyramid Panic JAR file.

1. Copy the directory path where you stored the JAR file as shown in the figure below.

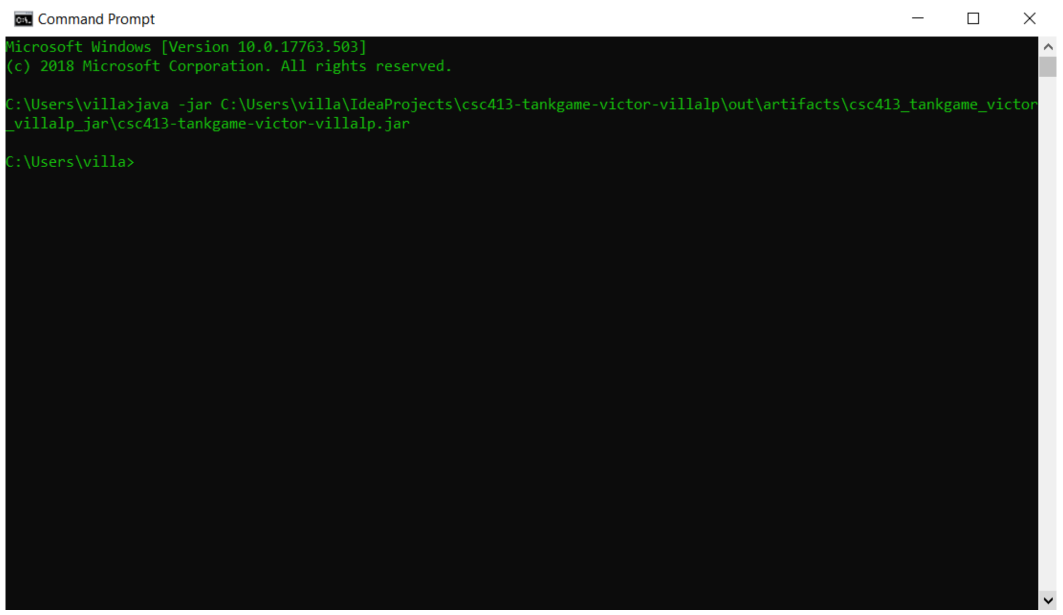

csc413-tankgame-victor-villalp [C:\Users\villa\
eWorld.java [csc413-tankgame-victor-villalp] - IntelliJ IDEA
File Edit View Navigate Code Analyze
csc413-tankgame-victor-villalp > out
csc413-tankgame-victor-villalp.jar >
Project
1: Project
csc413-tankgame-victor-villalp C:\Us
.idea
jar
META-INF
out
artifacts
csc413_tankgame_victor_villal
csc413-tankgame-victor-v
production
src
LICENSE
README.md
External Libraries
Scratches and Consoles
New
Cut Ctrl+X
Copy Ctrl+C
Copy Path Ctrl+Shift+C
Copy Relative Path Ctrl+Alt+Shift+C
Paste Ctrl+V
Find Usages Alt+F7
Analyze
Refactor
Add to Favorites
Delete... Delete
Run 'csc413-tankgame-vict...' Ctrl+Shift+F10
Debug 'csc413-tankgame-vict...'
Run 'csc413-tankgame-vict...' with Coverage
Create 'csc413-tankgame-vict...'...
Show in Explorer
Open in Terminal
Local History
Git
Synchronize 'csc413-tan...villalp.jar'
File Path Ctrl+Alt+F12
Compare With... Ctrl+D
Compare File with Editor
Z: Structure
Run:
GameWorld
2: Favorites

```
:ld extends JPanel {

inal int WORLD_WIDTH = 1920 ,WORLD_HE
final int SCREEN_WIDTH = 961, SCREEN_
dImage world;
jf;
 t2;
dImage backgroundImg;
dImage tankImg , lifeImg;
dImage bulletImg , smallExplosionImg,
dImage wallImg, breakableWallImg;
 player1Won = false;
 player2Won = false;
```

```
Process finished w
```

Figure 3.3-1. JAR File Directory

2. Open Command Prompt and type “java -jar” followed by the directory path of the JAR file and press **ENTER**.



```
Microsoft Windows [Version 10.0.17763.503]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\villa>java -jar C:\Users\villa\IdeaProjects\csc413-tankgame-victor-villalp\out\artifacts\csc413_tankgame_victor
_villalp_jar\csc413-tankgame-victor-villalp.jar

C:\Users\villa>
```

Figure 3.3-2. Command Prompt

3. The JAR file will as open a “Tank Wars” game window as shown in section 3.2 step nine.

4. Game Controls

4.1 Tank Wars Controls

The game controls for each player are specified below:

1. Player one can move and rotate across the map with the W, A, S, D keys (W: Move Forwards, A: Rotate left, S: Move Backwards, D: Rotate right) and shot bullets with the F key.
2. Player two can also move across the map with the arrow keys (UP: Move Forwards, LEFT: Rotate Left, DOWN: Move Backwards, RIGHT: Rotate Right) and shot bullets with the space bar.

1.4 Pyramid Panic Controls

The explorer game controls are specified below:

1. Use the arrow keys for movement (UP key: Move Forwards, LEFT key: Move Left, DOWN key: Move Backwards, RIGHT key: Move Right).
2. Press ENTER to shoot.

3. Activate scarab with the SHIFT Key.
4. Activate the sword's light with the SPACE key.

5. Assumption Made

In the implementation of both game I made the following assumptions:

1. A game menu, game control instructions and restart options are not required, but probably recommended.
2. Tank objects must rotate and move in angle.
3. Unlike tank objects, the player (explorer) in Pyramid Panic can only move in one direction at a time, meaning that the player cannot moved diagonally across the game map.

6. Tank Wars Class Diagram

The Tank Wars class diagram shown below illustrates the project's core abstraction. The abstract GameObjects class is template for the Bullet, PowerUp, Wall and Tank classes. All the GameObject child classes have the functionality of the GameObject class; they have a x and y coordinate, a Buffered image, and a rectangle object with the dimensions of the image positioned at the c and y coordinate. The child classes also implement a Collidable interface, which contains the method signature for detecting object collisions using rectangles. TankControls class gives an instance of a Tank functionality, while the GameWorld class is initializes and updates the status of the game.

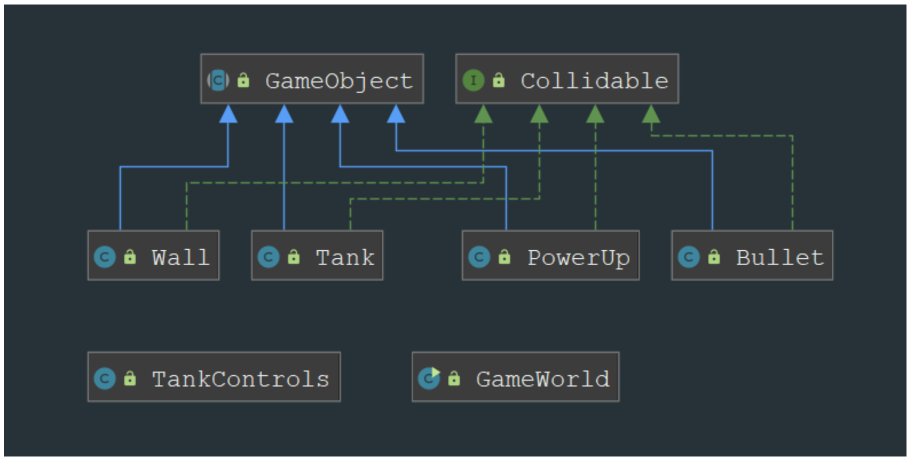


Figure 6.1-1. Tank Wars Class Diagram

7. Pyramid Panic Class Diagram

The Pyramid Panic class diagram shown in Figure 7.1-1 is shares the same project abstraction as the Tank Wars game. The abstract GameObjects class serves as template for the Beetle, Bullet, Explorer, MovableBlock, Mummy, PowerUp, Scorpion, Treasure, and Wall classes. All the GameObject child classes have the functionality of the GameObject class and implement the Collibable interface. The ExplorerControls class is almost identical to the TankControls class but includes two additional key controls. The GameWorld is also almost the same as the GameWorld class of the Tank Wars game, however, but there's an additional method to darkened the game screen when the player picks up for the sword.

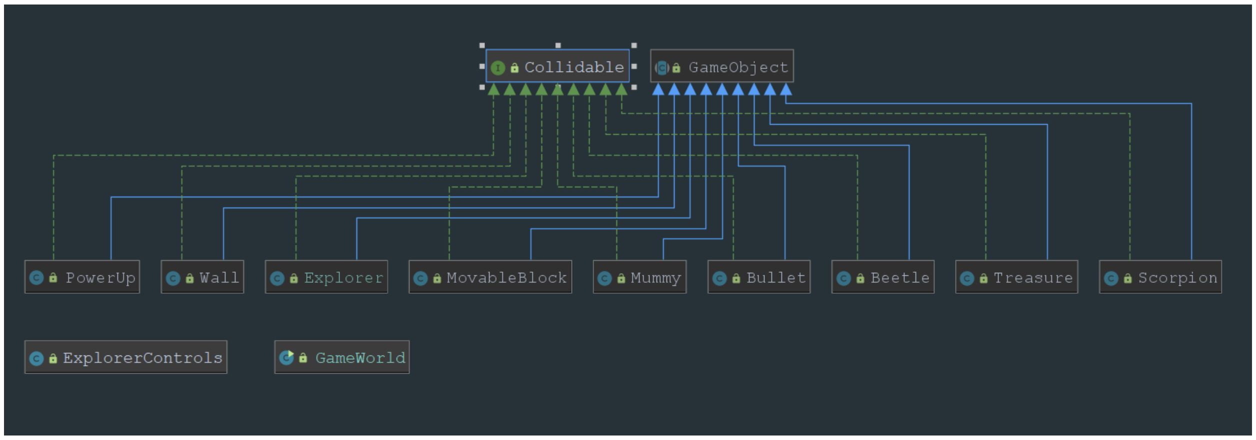


Figure 7.1-1. Pyramid Panic Class Diagram

8. Class Description of Shared Class

GameWorld

The GameWorld initializes the game and updates the status of the game and all instances of GameObjects. Both Tank Wars and Pyramid Panic GameWorld classes share the same functionality, but the class for Pyramid Panic an additional block of code was implemented to limit the visibility of the player when it wields the sword.

In this class, many variables and buffered images were declared and initialized, but they will not be discussed in detail. Rather, the functionality of key methods and code blocks will be briefly explained.

The init() method in this class initializes all the buffered images needed for each GameObject child class. Using a try catch block, the images are initialized from the images available in the resources folder. After initializing all the images, the Map.txt, made of numbers, is read using a BufferedReader. As the reader gets each line of the text file and parses them, objects are instantiated based on the number read from the file and then added to an ArrayList of type GameObject. For example, 1 and 2 were used to denote unbreakable and breakable walls, respectively, in both games.

The gameUpdate() method traversers the ArrayList<GameObject> with a for loop, determining whether the game objects is alive in order to call the update method specific to that object, otherwise the object is removed from the ArrayList.

The checkCollisions() method also traversers the ArrayList<GameObject>, but with two nested for loops to check if two objects have collided at any given time by determining if their rectangles intersect. Note that the second for loop starts at the next index of the to avoid checking for collisions with the same object. Also, the GameObjects must be casted to Collidable because not all GameObject child classes implement the interface.

Methods getScreenXCoord and getScreenYCoord take in a tank or explorer object to get the x and y coordinates of the objects in order to determine how to update the location of the sub images used to display a split screen in the tank game and follow the explorer in the second game.

Finally, the main class method initializes the game by calling the init() method and then updates the status of the game and the game objects by calling the gameUpdate(), checkCollision(), and paint() methods inside a forever loop.

GameObject

The GameObject class holds the main abstraction of both games. In this class, constructor GameObject passes x and y coordinate, a buffered image, creates a rectangle and initializes Boolean variable alive to true by default. There are getters and setters for all the variables being passed, except for the buffered image. Also, there are two abstract methods that are crucial for implementation of the game: update () and drawImage (). Each GameObject child class must override these abstract methods.

Bullet

The Bullet class extends GameObject and implements the Colliable interface. This class constructor calls the GameObject default constructor by passing a x and y coordinate and buffered image. The constructor also passes unique fields associated with a bullet. Horizontal and vertical velocity variables were initialized with constant speed using the angle of the tank and explorer to determine the direction of the bullet. If the bullet is alive the update() method updates the x and y coordinates of the bullet object and its rectangle location. The collision(Collidable) method set the default Boolean variable to false if the bullet object collides with an instance of any of the other GameObject child classes, except with power ups and treasure objects.

PowerUp

The PowerUp class extends GameObject and implements the Collidable interface. Its constructor creates a default instance of GameObject. In this class, the update() method is empty because no actions are required since power ups have not movement. However, the collision(Collidable) method sets the object's default Boolean variable alive to false when an instance of Tank and Explorer collides with the power up object. Lastly, the drawImage() method draws the buffered image associated with the object at a specified x and y coordinate.

Wall

The Wall class also extends GameObject and implements the Collidable interface. Its constructor also uses the GameObject default constructor but passes an additional Booelan variable to determine if the wall is set to be breakable in the Tank Wars or walk-through in Pyramid Panic. The update() method is empty because the location of the walls does not need to be updated. In the tank game, the collision(Collidable) method set the breakable wall's alive Boolean variable to false if it collides with a bullet. However, in the Pyramid game the collision method is empty because nothing changes in a wall object when collides with an object. Lastly, the drawImage(), methods draws the buffered image belonging to the object at x and y position.

Collidable

The Collidable interface is the shortest and simplest java class in both games but plays a crucial role. This interface implements the method signature of collision (Colldable), which is used by the classes that implement this interface to detect collisions between objects.

TankControls and ExpplorerControls

The TankControls and ExplorerControls classes enable control functionality for instance of the tank and explorer class, respectively. Both classes implement KeyLister interface and the keyTyped, keyPressed, and keyRealsed methods, which all take in a KeyEvent object. Their constructors pass integer key event values and their keyPressed and KeyPressed methods determined if a key was pressed or released by changing the value of Boolean variables associated with a key.

9. Class Description Specific to Tank Wars

Tank

The Tank class extends GameObject and implements the Collidable interface. Its constructor calls GameObject super, passes vx, vy, and angle variables and string variable which is used to determine if the tank object is player one or two. The update() method updates the location of the tank by calling the four different direction methods (moveFowards(), moveBackwards(), moveLeft, and moveRight()) and allows the tank to fire one bullet per second. The collision method checks collision between the object all instances of ther objects including another instance of tank. Lastly, in the drawImage() method the object was drawn with a green health bar, which decreased in length as the tank object took health damage.

10. Class Description Specific to Pyramid Panic

Beetle and Scorpion

The Beetle and Scorpion classes extend GameObject and implement the Collidable interface. Both classes implement the same functionality: objects move a constant speed, but if an explorer object is nearby the speed of the object doubles. However, beetles move up and down and scorpions left to right. The

Explorer

The Explorer class also extends GameObject and implement the Collidable interface. This class is implements similar methods of the Tank class. The update() method also updates the location of the explorer by calling our different direction methods (moveFowards(), moveBackwards(), moveLeft, and moveRight()) and shoot method, but calls to additional methods to activate scarabs and the sword. The collision method checks any collisions between the explorer object with any of the other object classes. For example, if an explorer only has one life remaining and it collides with an enemy the explorer will died. Lastly, the drawImage() method with the help of the runAnimation() method draws the walking animation of the exploere based on the direction in which its walking.

MovableBlock

The Movable class also extends GameObject and implement the Collidable interface. The class constructor calls GameObject super and passes a Boolean value to set the block to move horizontally or vertically. The update() method updates the location of the blocks and if the Boolean variable collide is true then the block is sent back to its initial position. The collide methods determines in which direction the block must in when the object collides with an explorer. If the block object collides with another movable block or a wall the Boolean variable collide is set to true, which is used to determine if the block should be sent to its original position.

Mummy

The Mummy class implements the functionality of the Beetle and Scorpion. In this class the, the constructor calls GameObject super and passes a two-dimensional array for images which are to be used for the walking animation of the mummy. The update() method calculates the distance away to the explorer and if the distance is less than 200 pixels the mummy will hunt the explorer. Now, if the mummy is scared it will walk away from the explorer for 10 seconds. The collidable method checks for collision between a mummy and any other game object, except treasure and power ups objects. The drawImage() method with the help of the runAnimation() method draws the walking animation of the mummy according to the direction in which is walking.

Treasure

The Treasure class extends GameObject and implement the Collidable interface. In this class, the constructor calls the default GameObject and passes four Boolean variables, which are used to determine if the treasure object is either an amulet, Anubis, scarab or a sword. The update() functions are empty because the location of treasure does not change. The collision method set the object Boolean variable alive to false if it collides with an explorer object.

11. Conclusion/Reflection

I had fun in making both the Tank Wars and Pyramid Panic game, even though I procrastinated and almost ended up not completing the games on time. This was the first time I ever made java games, but somehow, I made my games work. In this term project, I designed and implemented a working Tank Wars and Pyramid Panic game by following basic OOP principles and the Single Responsibility Principle (SRP).