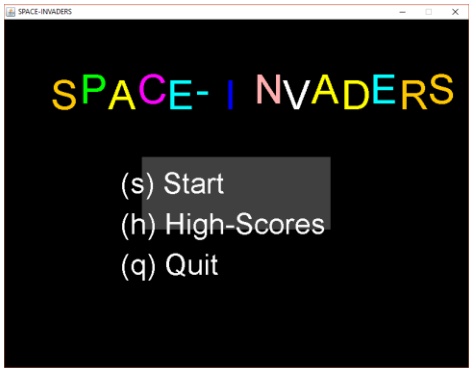


Name: Aviad Gottesman
User: Gottesa
ID: 311250484

## Classes:

**Ass7Game** – The main class of the project. Its job is to create the objects that the project will constantly use (GUI, GameFlow etc.).
It also creates the main menu and the tasks used to run the corresponding action.

**anime.AnimationRunner** – Its job is to receive a class implementing Animation and run his animation while *shouldStop* method return False.

**anime.KeyPressStoppableAnimation** – Its job is to receive a class implementing Animation and run his animation using AnimationRunner. It is responsible of running the animation until a specified button is pressed.

**anime.CountdownAnimation** – Its job is to show a countdown from a specified number till zero in a specified number of seconds. This class implements Animation and is using AnimationRunner to run.

**anime.EndScreen** – Its job is to show a end screen when the player loses. This class implements Animation and is using AnimationRunner and KeyPressStoppableAnimation to run.

**anime.HighScoresAnimation** – Its job is to show a high scores screen, called automatically in the end of a game and manually if the user chooses it from the menu. This class implements Animation and is using AnimationRunner and KeyPressStoppableAnimation to run.

**anime.LevelIndicator -** Its job is to show the level name during the gameplay. This class implements Sprite and runs by the GameLevel class.

**anime.LivesIndicator** – Its job is to show the number of lives left for the user in a current game. This class implements Sprite and runs (draws onto screen) by the GameLevel class.

**anime.ScoreIndicator** – Its job is to show the score for the user in a current game. This class implements Sprite and runs by the GameLevel class.

**anime.PauseScreen** – Its job is to show a pause screen after the user pressed the `p' button. This class implements Animation and is using AnimationRunner and KeyPressStoppableAnimation to run.

**anime.MenuAnimation<T>** - This is a generic class; his job is to represent and show a menu. It is implementing Menu<T> (an interface that extends Animation), using AnimationRunner to run.

**gameobjects.Ball** – Its job is to represent a bullet in the game. This class implements Sprite in order to be able to draw himself onto the game screen, and HitNotifier in order to be able to notify other object we collided with about the collision.

**gameobjects.BallRemover** – Its job is to perform an action after a collision happened, he is responsible for removing the ball from the game if he reached outside of the screen. implements HitListener.

**gameobjects.Block** – It implements Collidable so we can collide with him, Sprite so could draw him and HitNotifier so will be able to notify objects we collided with. It represents a block in the game, used to create the shields.

**gameobjects.BlockRemover** – Its job is to perform an action after a collision happened, he is responsible for removing the ball and the alien after a bullet hit's an alien from the game. implements HitListener.

**gameobject.CollisionInfo** - Its job is to hold the information about a collision (the Point of the collision and the Collidable object that was hit).

**gameobject.Counter** – Its job is to hold a number representing something in the game (number of blocks, number of lives etc.) and he is responsible of holding the data and update him when asked.

**gameobject.GameEnvironment –** Holds a list of Collidable objects; It is responsible of updating this list and when a collision occurs tell us what is the which object from the list was the first to be hit.

**gameobject.GameFlow** – Its job is to receive a LevelInformation object from main that represent a game level and run it using GameLevel class. It is responsible for keeping track of lives, score and high scores for each game.

**gameobject.GameLevel** – Its job is to run a level that was received from GameFlow, initialize it and run it. It is responsible of running each frame (This class implements Animation and is using AnimationRunner to run a frame) and call each Sprite *timePassed* method thus telling each object to do a certain action/check (i.e. tell ball to move and check for collisions). It is also responsible for stopping his animation when the user win\looses.

**gameobject.MenuSelection<T>** - A generic class, his job is to hold a single menu's option.

**gameobject.Paddle** – Its job is to represent the player's spaceship. This class implements Sprite and runs (draws onto screen) by the GameLevel class; Also implements Collidable so he will collide with bullets.

**gameobject.ScoreTrackingListener** - Implements HitListener, his job is to keep track of the score (hold the information and update it) using a Counter.

**gameobject.SpriteCollection –** Holds a list of Sprite objects; It is responsible of updating this list and when ordered to call each object *timePassed* and *drawOn* methods.

**gameobject.Velocity** – Its job is to represent a Ball velocity.

**geometry.Rectangle** - Its job is to represent a rectangle (a points ,height and width).

**geometry.ColoredRectangle** - Its job is to represent a rectangle with a color.

**geometry.Line** - Its job is to represent a line (two points).

**geometry.Point** - Its job is to represent a point (an X and a Y coordinates).

**level.Level** - Implements LevelInformation, his job is to return the properties of a level.

**saves.ScoreInfo** – Holds the information of a single high score (score and name).

**saves.HighScoresTable** – Holds a list of ScoreInfo of a pre-determined size representing the high scores. Its responsibility is to update this list and save/load for each change to a file.

**saves.SortByScore** - Implements Comparator<ScoreInfo> used to sort the HighScoresTable.

**enemies.Alien** – It extends Block, so we can collide with him, draw him, and able to notify objects we collided with. It represents an alien in the game, so he is also able to shoot and move unlike Block.

**enemies.AlienRow** – Represent a row (|) of aliens, implements Sprite so we can draw the aliens. It is responsible of moving the aliens in its row, tell an alien to shoot, and update his list of aliens.

**enemies.AlienCol**– Represent a column (-) of AlienRow, , implements Sprite so we can draw the aliens. It is responsible for:

- Check each *timePassed* if we should shoot, if so select a random raw and tell it to shoot.
- Check each *timePassed* if we should change direction/speed of the formation.
- Draw all the aliens by calling each row *drawOn* method.
- Reset the formation when the player starts his second life.

**interfaces.Animation** – An interface for an animation, has the following methods:

- *void doOneFrame(DrawSurface d, double dt)* { that draws the object }
- *Boolean shouldStop()* { checks if we should stop }

**interfaces.Collidable** - An interface for an object we can collide with, has the following methods:

- *Rectangle getCollisionRectangle()* { Return the Rectangle that was hit.}
- Velocity hit(Ball hitter, Point collisionPoint, Velocity currentVelocity) {
  Notify the object that we collided with it at collisionPoint with a given velocity.
  The return is the new velocity expected after the hit.}
- *void removeFromGame(GameLevel gameLevel)* {
  Removing the Collidable from the game.}
- *boolean isFriend()* { Tells if the object is friendly to the player.}

**interfaces.HitListener** - An interface for an object that need to do an action when a hit occurs, has the following methods:

- *void hitEvent(Collidable beingHit, Ball hitter)* {
  Does a specific action according to the implementation.}

**interfaces.HitNotifier**- An interface for an object that need to do notify when a hit occurs, has the following methods:

- *void addHitListener(HitListener hl)* { Adds a HitListener to the objects listeners list.}
- *void removeHitListener(HitListener hl)* { Remove a HitListener to the objects listeners list.}

**interface.LevelInformation** - An interface for an object that represent a level, has the following methods:

- *int paddleSpeed()* { Returns the Paddle's speed.}
- *int paddleWidth()* {Returns the Paddle's width.}
- *String LevelName()* {Returns the level's name.}
- *Sprite getBackground()* {Returns the level's background's Sprite.}
- *AliecnCol blocks()* {Returns the AlienCol's of the level.}
- *int numberOfBlocksToRemove()* {Returns the number of aliens we need to kill to advance.}

**interface.Menu** - An interface for an object that represent a menu, has the following methods:

- *void addSelection(String key, String message, T returnVal)* {Adds a selection to menu. }
- *T getStatus()* {Returns the selected value.}

**interface.Sprite** - An interface for an object that has the ability to draw himself on the screen, has the following methods:

- *void drawOn(DrawSurface d)* { raw the Sprite to the screen.}
- *void timePassed(double dt)* { Notify the sprite that time has passed.}

**interface.Task** - An interface for an object that has the ability to run a specific code, has the following methods:

- *T run()* {Runs a specific code (according to the implementation.)}

## A ‘brief’ description of how you implemented the following:

### The Aliens formation:

I created 3 classes to handle the alien’s formation: Alien, AlienRow and AlienCol. Alien is a class for a single Alien, AlienRow is a list of Aliens and AlienCol is a list of AlienRows. In the Sprite list I added only the AlienCol, so every frame we call it’s *timePassed* method which start’s the formation movement in the following way:
AlienCol has this members:

```
private static double moveDis = 1; // How many pixels to move.
private static double moveDir = 1; // To which side to move (if >0 to the right, else left)
```

1. Check if any AlienRow got to one of the edges of the GUI, if so:
- Move each AlienRow down.
- Set moveDis *= 1.1; // raising the move speed.
- Set *= -1; // change to move direction.

2. Move all the AlienRow a (moveDis*moveDir) pixels.

### **The shields**

I created the shields as part of the GameLevel *initialize method, it creates 3 rectangles each Is built by 50*3 Blocks. I add those Blocks to the following lists:*
*- The Sprite list so we will draw them.*
*- The Collidable list so we can collide with them.*
- *We add a HitListner* BlockRemover so they will be destroyed by a Ball.
- A private member `List<Block> shieldsList` used to keep track of them, with this list we can keep the same shields when user dies or delete all of them and create new of the user wins.

## Shots by aliens

I created 3 classes to handle the alien's formation: Alien, AlienRow and AlienCol. Alien is a class for a single Alien, AlienRow is a list of Aliens and AlienCol is a list of AlienRows. In the Sprite list I added only the AlienCol, so every frame we call it's *timePassed* method which start's the shooting in the following way:

- I created 3 private members:

```
private List<AliensRow> aliensColList = new ArrayList<>(); // A list of AlienRows.
private static long nextShot = java.lang.System.currentTimeMillis(); // The next time to shoot.
private static final long SHOT_INTERVAL = 500; // the shooting interval.
```

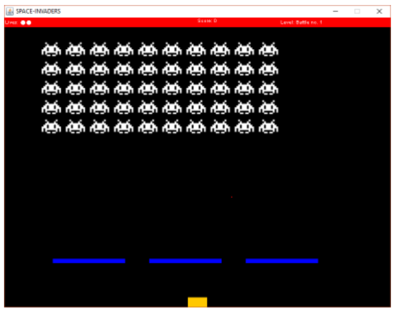
- if the current time is passed nextShot:

sets nextShot = current time + SHOT_INTERVAL.

Choose a random number between [0, aliensColList.size()] and call the random-result-index *shoot()* method. It in turn finds the lowest Alien in the row and calls his *shoot()* method. Which finally shoots by creating a Ball below him with:

1. boolean friendly = false // A boolean used to indicate if an object is friendly to the player.
2. the GameLevel to be at.

I pass this Ball to his public method *shoot() which according to his affiliations, sets his color, his velocity, and speed. Than adds him to the GameLevel.*



## Shots by player

I have the following members in Paddle:

```
private static long nextShotTime; // The next time to shoot.
private static boolean firstShot = true; // A check if this is the first shot.
private static final int SHOOT_INTERVAL = 350; // the shooting interval.
```

In the *timePassed()* method of Paddle I added a check if SPACE was pressed, if so- check if we are allowed to shoot (firstShot OR passed nextShotTime) if so- sets firstShot and nextShotTime, shoots by creating a Ball above him with:

1. boolean friendly = true // A boolean used to indicate if an object is friendly to the player.
2. the GameLevel to be at.

I pass this Ball to his public method *shoot() which according to his affiliation, sets his color, his velocity, and his speed. Than adds him to the GameLevel.*

## Some loose ends

In *hitEvent* method of BlockRemover I check the friendliness of the hitter (Ball) and the object being hit (Block) and do the following:
- If both are friendly (we hit the shields), destroy the Block.
- If Ball is friendly but Block isn't, destroy Block and subtract 1 from the Block Counter of the current GameLevel.
- if Ball is not friendly but Block is (Alien bullet), destroy the Block.
- In any case destroy the Ball (remove from game).