**List of interfaces:**

*Animation***-** An animation runs something on the screen of the game until stopping condition.

*Menu-* A menu has different selections and a key for each one.

*Task-* A task is in charge of running something.

*HitListener-* A hit listener object is notified when a hithas been occurred.

*HitNotifier-* A hit notifier object notify all his listener that a hit has been occurred.

*LevelInformation-* Set and get all the parameters of a level.

*Background-* It Draws the background of an object.

*BlockCreator- Create a block at the specified location.*

*Collidable-* The Collidable interface is used by objects that can be collided with.

*Sprite-* All objects used in the game are sprites.

**List of classes:**

*Line-* Find intersection points between two lines. The collision between the ball and an object is been checking here, when one line is the trajectory of the ball to the object, and the second is when one line is the trajectory of the ball to the object, and the second is the outline of the object.

*Point-* Has a x and a y coordinate. Every geometry object has a location which is a point.

*Rectangle-* Has a point, a width and a height. It is the shape of different objects in the game.

*Block-* Has a *Rectangle* shape and a color. The ball can collides it in the game.

*Alien-* Extends the *Block,* has a number of column and row and can change its location.

*Ball-* Has a *Point* center which is its location, a size and a velocity. The ball moves in the game and changes its velocity when it hits objects.

*Paddle-* In this game the *Paddle* is the space ship that shoot bullets to the aliens and can move right or left.

*AnimationRunner-* Takes an animation object and runs it.

*CountDownAnimation-* Countdown from 3 to 1, it will show up at the beginning of each turn.

*Counter-* Count things, can increase or decrease the count and get the current count.

*EndScreen-* An animation at the end of the game, shows the game over screen and the score.

*HighScoresAnimation-* An animation that shows the table of high scores.

*KeyPressStoppableAnimation-* An animation that runs other animations until a specific key is

pressed.

*MenuAnimation-* An animation that shows the menu on the screen with it selections.

*PauseScreen-* An animation of a screen that is showed when the player pressed the pause key.

*GameEnvironment-* In charge of getting the collisions, adding or removing a collidble object.

*GameFlow-* Builds the menu and run it.

*GameLevel-* Initialization of all object in the game, calling methods which are in charge of

moving the game, receiving the information about the number of remaining lives, aliens...

*ShowHiScoreTask-* A task that runs the high scores animation.

*StartGameTask-* Run in a loop the *Gamelevel* until there is no more lives and at the end of the

game, opens the score dialog (if high score), shows the end screen and then the high score

screen.

*BlocksDefinitionReader-* Read the block\_definitions file and creates blocks by the given parameters.

*BlocksFromSymbolsFactory-* Associate blocks and spacers to their symbols.

*ColorBackground-* Type of *Background,* receive a rectangle and the surface, and draw its background by the given color.

*ColorParser*- Has a map of colors name and their corresponding color.

*CreateLevel-* Implement the *LevelInformation,* is initializes the level with the parameters in the level definitions file.

*ImageBackGround-* Type of *Background,* receive a rectangle and the surface, and draw its background by the given image.

*LevelBackgroundSprite-* In charge of the level’s background which is a field in the

*LevelInformation, d*raws the color and the name of the level.

*LevelSpecificationReader-* Read the level\_definitions file and creates the level by the given parameters.

*AlienRemover-* Is a hit listener. Each alien adds it when created and when there is a hit, the alien and the hitter ball are removed from the game.

*BallRemover-* Is a hit listener. Two blocks are created, one above the screen and one below and both add the *BallRemover* as a listener, so that when the ball goes out of the screen, it is removed from the game.

*BlockRemover-* Is a hit listener, the same as the *AlienRemover* but for the blocks in the shields.

*HighScoreTable-* In charge of reading the high scores file, loading it and adding scores to it.

*ScoreInfo-* An object that keeps the name and the score of a player. Each line in the high scores file is a *ScoreInfo.*

*alienMover-* Holds a list of all the aliens and is responsible for moving them to the left, to the right, to the next line when they hit the side borders and bringing them back to the top.

*ColisionInfo-* Has an object that the ball collided it and the collision *Point*.

*LivesIndicator-* Is a *Sprite,* it shows the number of remaining lives on the screen.

*ScoreIndictor-* Is a *Sprite,* it shows the current score on the screen.

*SpriteCollection-* Holds a list of all the sprites objects, goes over them and call their functions.

*Velocity-* Holdsthe dx and dy values of the ball velocity. Can computes those values by giving an angle and a speed.

*Ass7Game-* The main of the game which runs the program.

**Implementation of alien formation (a):**

I created an *Alien* class which expands the block class. I added the methods moveRight (), moveLeft () and moveLine () that change the position of the aliens accordingly. I also created an *AlienMover* class which holds a collection of all the aliens and has the same methods as the *Alien* class (and others). The moveRight () and moveLeft () methods of the *AlienMover* check if the rightest or leftest column will hit the borders. If they will, the methods call the moveLine () method which checks if the aliens can go down or have reached the shields. Each of those methods (after checking that the aliens can move) goes over each alien and calls its corresponding method (moveLeft/Right/Down). The *AlienMover* constructor receive the x coordinate of the original upper-left alien and the gap between the aliens. It also keeps a member which count the number of rows that the aliens have gone down. When the aliens reach the shields, *AlienMover* brings them back to the top. The x coordinate of each alien changes: the x of the first to fifth alien changes to the original x, the x of the sixth to tenth changes to the original x + the width of the alien + the gap, and so on (the aliens are created by column). The y coordinate is the current y when the ‘num of rows down’ \* (the height of the alien + the gap) is reduced from it.

**The shields (b):**

The shields are made of little blocks when their sizes and color are specified in the

level\_definitions file. Those blocks are the same blocks as in Arknoid, except that when a ball

hits them, the ball doesn’t change it direction but is removed from game.

**The shots by the aliens or by the player (c, d):**

I added two members (double) to the *GameLevel* class, which are the remaining time before the player can shoot and the remaining time before the alien can shoot. Those members are initialized at 0.35 and 0.5 respectively. In the doOneFrame () method, the ‘dt’ value is reduced from those members and when there is no more remaining time (their values < 0), the bullets are created. The bullets are regular balls, with an angle of 0 (bullet of player) or 180 (aliens bullet), and after they are created, the remaining times are again 0.35 or 0.5.