JAVA 1

Space Pig Fighter

Authors : Vincent Reynaert, Nicolas Sobczak

Contents

Chapter 1

Game presentation

1.1 What is it ?

"Space Pig Fighter" is a game that is played in the terminal by 2 players. Each player is a space pig and have to beat the other one.

A game happens in 2 phases. The first one is a spacecraft battle. The second one is a melee battle. Each spacecraft has several caracteristics. Each pig has several caracteristics and some weapon.

1.2 Rules

1.2.1 Animal class

Here are the concept we chose :

Animal class	Life	Force	Resistance	Special attack
Bear	mid	mid	big	damageAnnulation
Chicken	low	big	mid	triple attack
Duck	big	mid	low	fly
Pig	mid	low	big	moreDamage
Tiger	mid	big	low	paralyze foe which can’t attack next turn

Here are the exact values we chose :

Animal class	Life (hp)	Force	Resistance	Special attack
Bear	1000	110	40	damageAnnulation: nn
Chicken	800	130	20	triple attack: nn
Duck	1200	110	0	fly: nn
Pig	1000	90	40	moreDamage: nn
Tiger	1000	130	0	paralyze foe which can’t attack next turn: nn

1.2.2 Animal specialAttack

Bear - damageAnnulation

Pig - moreDamage

Tiger - paralyze foe which can't attack next turn

Chicken - tripleAttack, 1 turn to charge

Duck - fly, dodge current attack and attack next turn

1.2.3 Meteorites malus

Size	Malus
small	-20 hp
medium	-50 hp
big	-100 hp

1.2.4 Stuff choice

You have 2 skill points to share between offensif and defensif stuff. You may choose to boost your attack at the expense of the your defense or to boost your defense at the expense of the your attack. Unless you prefer to choose a well balanced build.

Build	Attack points	Defense points
Offensive	2	0
Well balanced	1	1
Defensive	0	2

Here are the bonus value of each stuff :

Build	Offensive stuff	Stats bonus	Defensive stuff	Stats bonus
Offensive	Axe	40	None	00
Well balanced	Sword	20	Helmet	20
Defensive	None	00	Shield	40

1.3 How it is thought/programmed

Each player plays when it is its turn.

Chapter 2

Story: what happens when you launch the game

What happens when you launch the game ?

2.1 Start the game

Game welcome players.

- player 1 is invited to choose his animal, enter animal's pseudo and color (pink by default), his spacecraft's color (gray by default).
- player 2 is invited to choose his animal, enter animal's pseudo and color (pink by default), his spacecraft's color (gray by default).

2.2 Part 1 of the game

- launch part1 of the game: space battle. You have to find the right location of the other player's spacecraft by entering a position. Each player try to guess turn by turn. You have to be careful, avoid meteorites ! Otherwise your pig's life will decrease.
- when a player find the other one's spacecraft, he climbs aboard and it's time for part 2 of the game.

2.3 Part 2 of the game

Players are welcomed to choose a stuff build in order to fight the other player.

1 turn happens in 3 steps:

1- Player 1 choose an action for his animal to do (choose between normal attack, special action and scream)

2- Player 2 choose an action for his animal to do (choose between normal attack, special action and scream)

3- Resolution

Game is over when a animal has no life point left. Since the resolution happens after both player's action, the result can be a draw.

2.4 End

At the end of the game, a file is created with the game summary written in it. If the file already exists, it is overwritten.

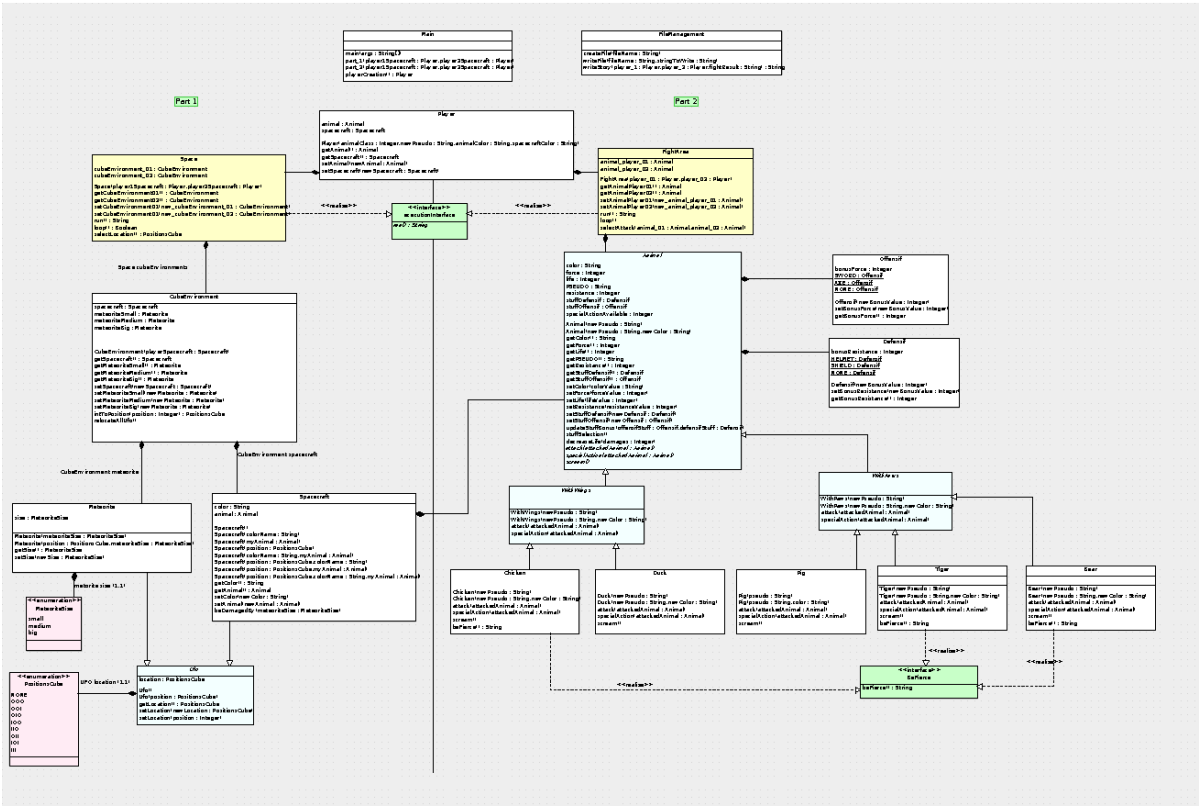
Chapter 3

Development part

Each player plays when it is its turn.

3.1 UML

Here is the global UML diagram of the program:



Since you can't see anything on this screenshot, there bigger screenshots below.

Blue classes are abstract classes.
Green classes are interface.
Pink classes are enumeration.
Yellow classes are the two main classes from the 2 different parts of the game.

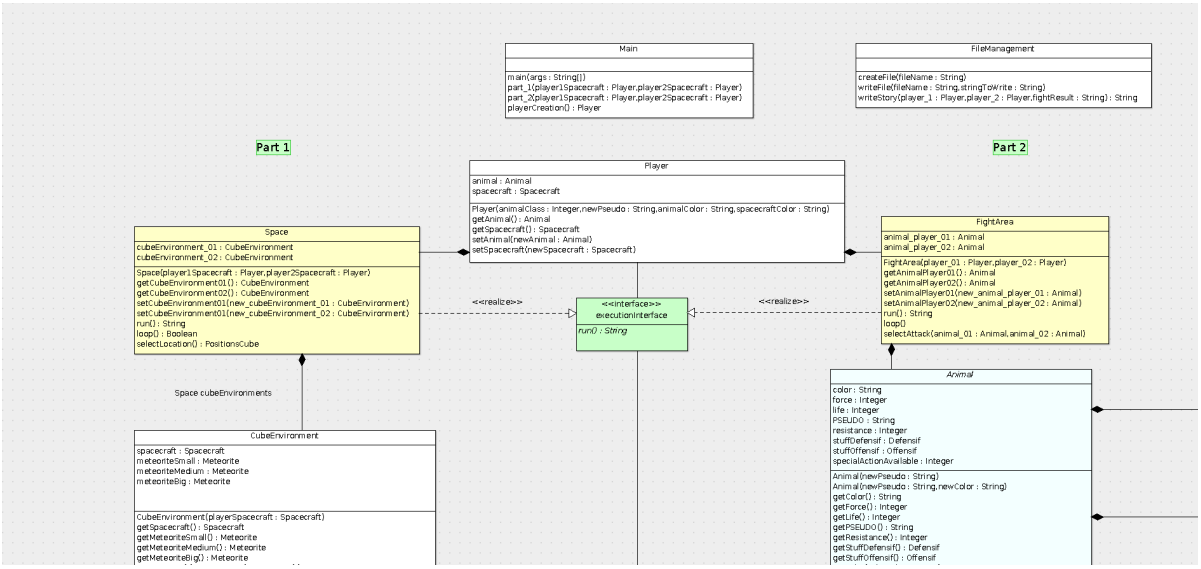


Figure 3.1 left screenshot 1

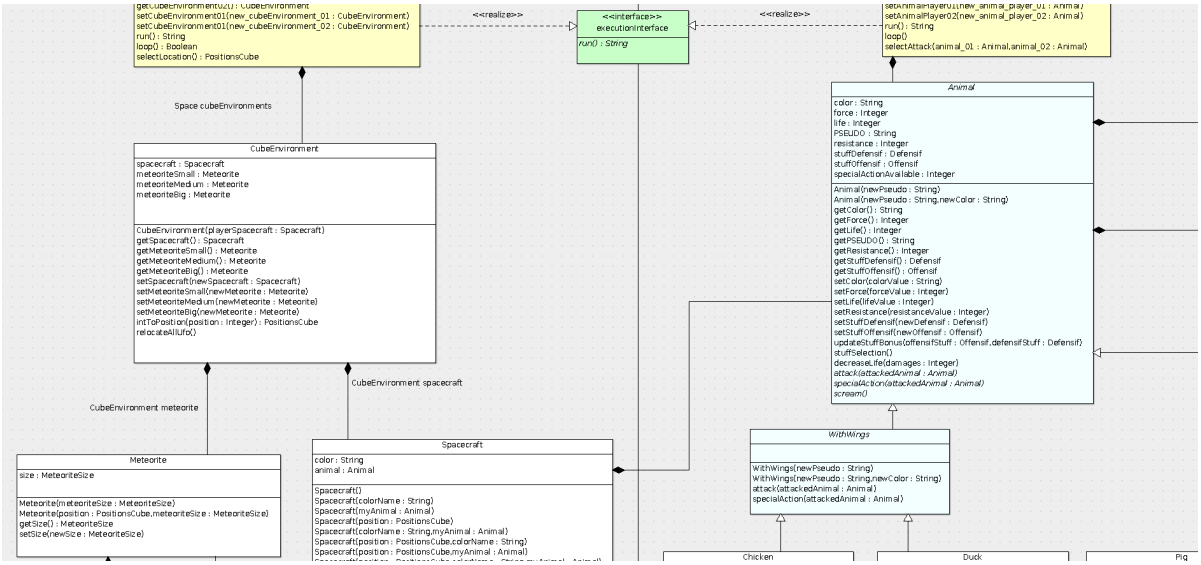


Figure 3.2 left screenshot 2

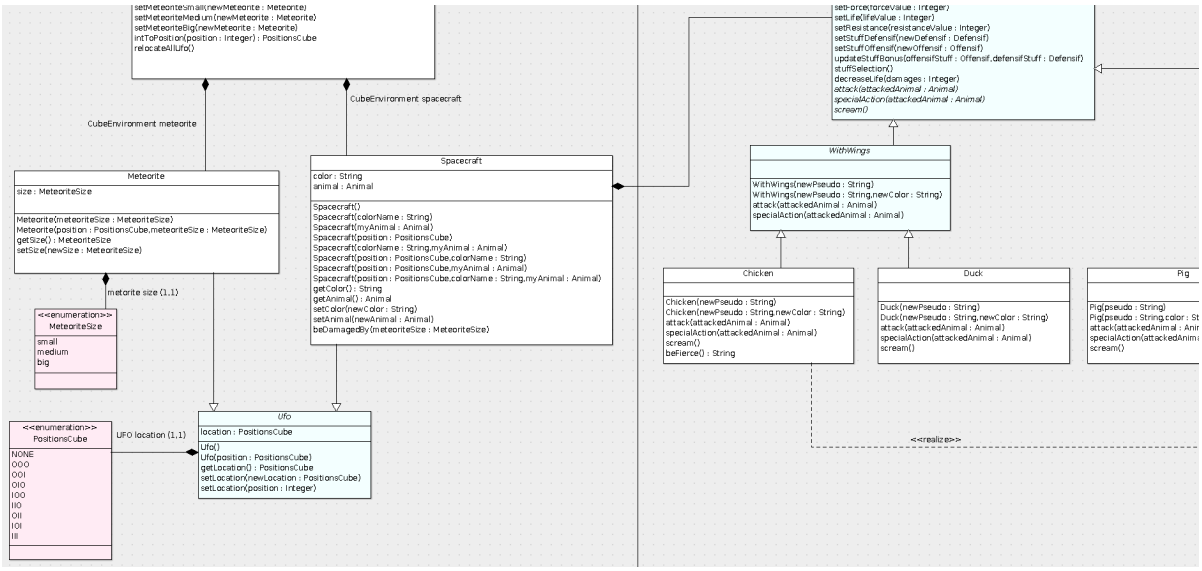


Figure 3.3 left screenshot 3

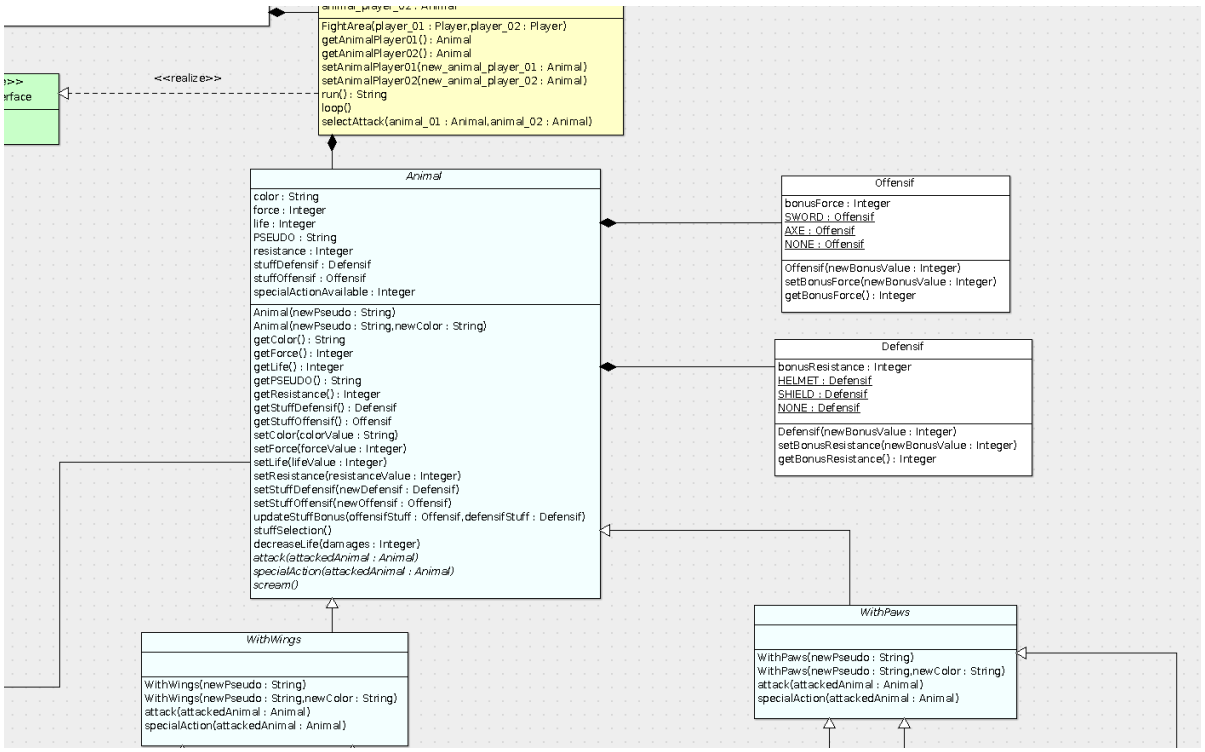


Figure 3.4 right screenshot 1

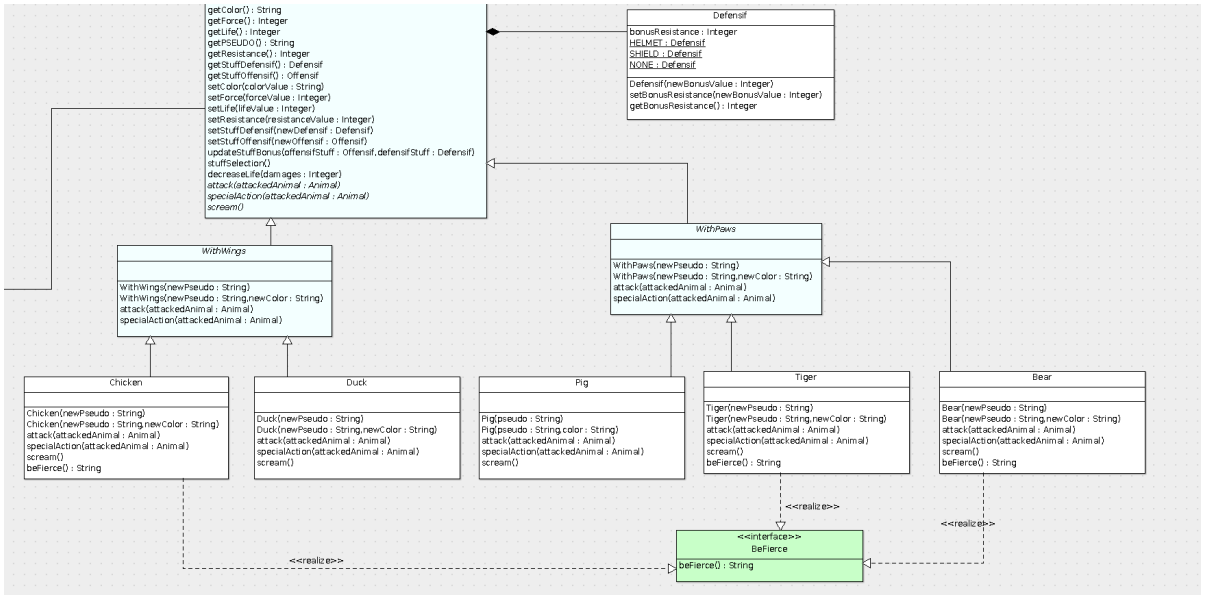


Figure 3.5 right screenshot 2

3.2 Technical part: class description

<insert here: brief class description>

3.2.1 Main

This class contains the main functions:

- main : main function that calls all the following functions.
- playerCreation : function that create the 2 players.
- part_1 : function that runs game part 1.
- part_2 : function that runs game part 2.

3.2.2 FileManagement

This class contains all useful functions to save the game story in a file. We chose to put them in a class in order not to overload the Main class.

3.2.3 Player

We created a Player class that keeps all information about each player. That is to say that a player contains a spacecraft and its animal. It is from this class that we can access all information at any time and everywhere in our code.

3.2.4 The 2 main classes of the game

We created 1 class for each part of the game. It is from these 2 classes that each part is run. They both implements the executionInterface interface.

3.2.4.1 executionInterface interface:

This interface has only one function: *run()*. We decided to create this interface in order to create a name convention for the function which runs each part of the game. By doing this, the Main class won't change, it will always call the *run()* function of each class even if each class change.

3.2.4.2 Space class:

It is composed by 2 CubeEnvironments created thanks to the 2 Players. It has 3 main functions :

- run() : main function from the interface, it runs all game part 1.
- loop() : it runs the main loop while no spacecraft has been found, each player select a location en try to guess spacecraft postition.
- selectLocation(): it return the position selected by a player.

3.2.4.3 FightArea class:

It is composed by 2 Animals created thanks to the 2 Players and a list of special actions. It has 4 main functions :

- run() : main function from the interface, it runs all game part 2.
- loop() : it runs the main loop while no dead animal has been found, each player select an action to do.
- selectAttack(): it allows a player to select an action for its animal to do.
- solveRound(): this function manage special actions.

3.2.5 Part1

3.2.5.1 CubeEnvironment class

We thought the space environment in a particular way. Indeed, we assimilate it to 2 cubes, 1 for each player. That's why the Space class is composed of 2 CubeEnvironment. Each cube is composed of a spacecraft and 3 meteorites. They can be located to 8 different positions that correspond to each corner of the cube.

During the 1st part of the game, each player try to find the location of the other one's spacecraft. Of course he has to avoid meteorites that decrease the life. Once one player find the other one, part 2 of the game is started.

3.2.5.2 UFO class

It is an abstract class. It was created in order to manage position of both meteorites en spacecrafts. That's why Meteorite class and Spacecraft class both extends UFO abstract class.

To manage location, an UFO has an attribute *location*. We also created function which make us be able to manage location. Constructor was overloaded in order to create a UFO default position (000) or take the position in parameter.

3.2.5.3 PositionsCube enumeration

This enumeration enumerates all available positions in a cube. These positions match each corner of the cube. They are coordinates.

3.2.5.4 meteorites

There are 3 meteorites in each cube. A Meteorite has size which can be one from the MeteoriteSize enumeration. The size impact the amount of life to withdraw to an animal if a player collides a meteorite.

3.2.5.5 MeteoriteSize

This enumeration enumerates all existing meteorite size.

3.2.5.6 spacecraft

There is one spacecraft in each cube. Spacecraft class has a color and an Animal. The spacecraft can be damaged by a meteorite. A damaged spacecraft means its animal life decreases.

3.2.6 Part2

3.2.6.1 Animal class

It is an abstract class.

3.2.6.2 WithWings class

It is an abstract class which extends animal class. It overrides *attack()* function to caracterize it by the way the animal attack (with paws or with wings).

3.2.6.3 WithPaws class

It is an abstract class which extends animal class. It overrides *attack()* function to caracterize it by the way the animal attack (with paws or with wings).

3.2.6.4 Bear class

Bear is an animal with paws. That's why it extends WithPaws abstract class. It overrides *attack()*, *specialAction()* and *scream()* functions. Since Bear is a fierce animal, it implements BeFierce interface and overrides *beFierce()* function.

3.2.6.5 Chicken class

Chicken is an animal with paws. That's why it extends WithWings abstract class. It overrides *attack()*, *specialAction()* and *scream()* functions. Since Chicken is a fierce animal, it implements BeFierce interface and overrides *beFierce()* function.

3.2.6.6 Duck class

Duck is an animal with paws. That's why it extends WithWings abstract class. It overrides *attack()*, *specialAction()* and *scream()* functions.

3.2.6.7 Pig class

Pig is an animal with paws. That's why it extends WithPaws abstract class. It overrides *attack()*, *specialAction()* and *scream()* functions.

3.2.6.8 Tiger class

Tiger is an animal with paws. That's why it extends WithPaws abstract class. It overrides *attack()*, *specialAction()* and *scream()* functions. Since Tiger is a fierce animal, it implements BeFierce interface and overrides *beFierce()* function.

3.2.6.9 BeFierce interface

This interface was created to caracterize scream of some animals that are said to be fierce. It contains 1 function, *beFierce()* function.

3.2.6.10 Offensif class

Each animal has an offensive stuff which gives it a force bonus. Offensif class is here to do that. It has force bonus value and constants that defines existing offensive stuff.

3.2.6.11 Defensif class

Each animal has an defensive stuff which gives it a force bonus. Defensif class is here to do that. It has force bonus value and constants that defines existing defensive stuff.

3.2.7 Set the game

- set Player class for each player.
- set Space class with 2 CubeEnvironment (1 for each player). Each CubeEnvironment is set with 3 meteorites and 1 spacecraft.
- set FightArea class with 2 pigs. Each pig is initialized with stuff selected by the player.

3.3 Encountered difficulties

3.3.1 Special action

Special actions are very different. So we had to think our code so that it would be able to welcome each special action. We had to modify our code.

3.3.2 Exception

We created an exception. We had difficultie because it was the first time and we didn't undertand exception very well. We no longer do !

Chapter 4

Conclusion

We think our project cover lots of different aspect of java language. Besides we enjoyed doing this game. That's why we may add a graphical interface in the future.