BEN HAMOUDA Amel
HARDY Elise
RUBIO Julien

UNIVERSITÉ PARIS-EST MARNE-LA-VALLÉE

Developer Guide
Wall-J : The Space Cleaner

Java Object programming

BEN HAMOUDA Amel HARDY Elise RUBIO Julien

Group 3

BEN HAMOUDA Amel
HARDY Elise
RUBIO Julien

Summary

BEN HAMOUDA Amel
HARDY Elise
RUBIO Julien

I. Objective of the program

Wall-J : the Space Cleaner is a directed game object written in JAVA.
This game aims to clean the game area of present waste by pushing them with bombs into the trashcans located over the screen.
The user will lead a small robot (Wall-j) who can put three bombs.
Exploding the bombs will push the waste to the trash can.
A waste disappears when it touch a trash can.
The level is finished when all the waste has been cleared, the next level is loaded if it exists.

II. Program development

A. Classes

For this project we use 13 classes (including 2 abstract class) and 2 interface:

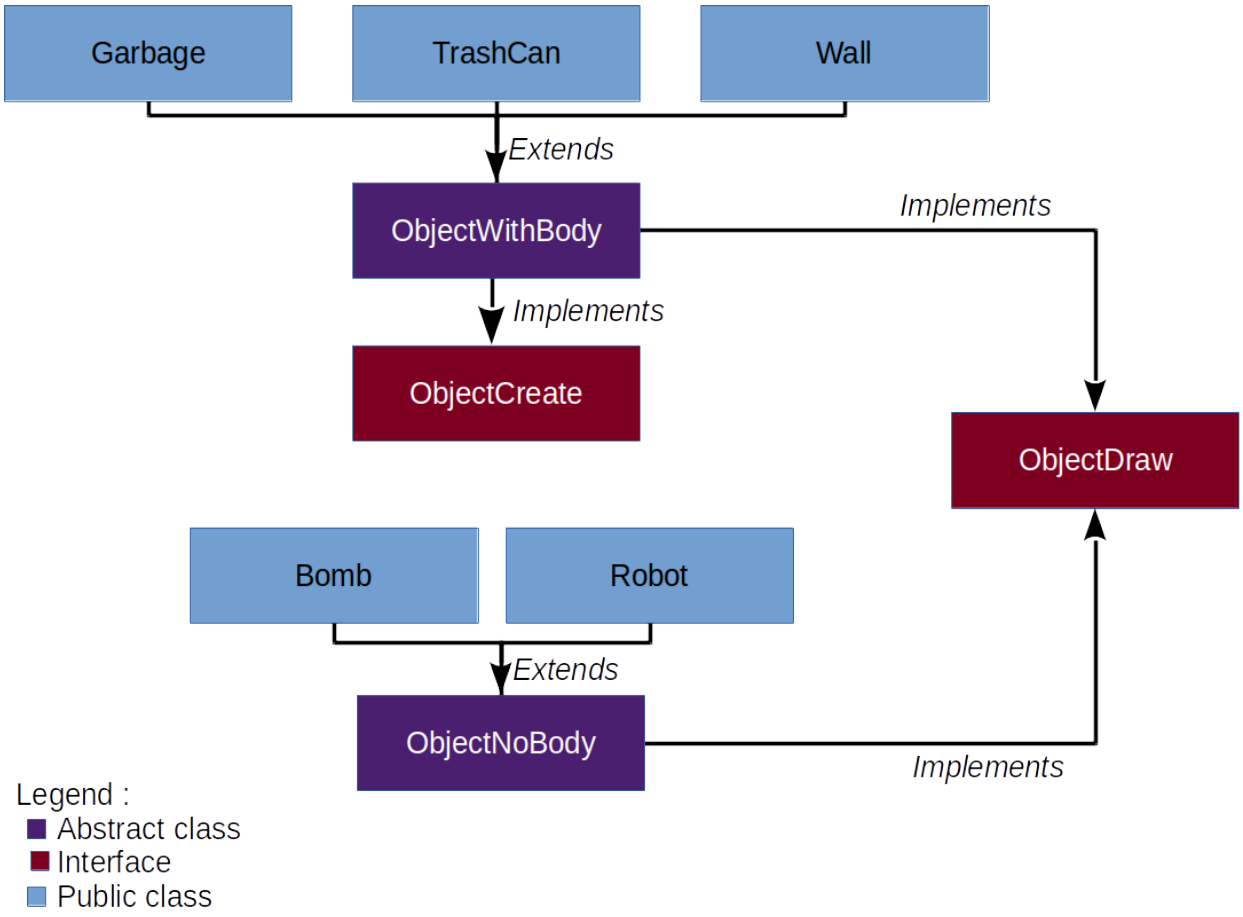
- → LevelLoader : use to read files and load the levels.
- → The Objects:
  - → Wall : represent a wall.
  - → TrashCan : represent a trash can.
  - → Garbage : represent a garbage.
  - → Robot : represent the robot wall-j.
  - → Bomb : represent a bomb.
  - → Point : represent a point.
  - → AStar : the algorithm to move the robot.
  - → Area : represent the game area and include the main.
- → The Interfaces:
  - → ObjectCreate : to create a object .
  - → ObjectDraw : to draw a object.
- → The Abstract Classes:
  - → ObjectNoBody : represent a object without body (no physical interation).
  - → ObjectWithBody : represent a object with body (with physical interation).

→ The others Class:

→ Collision : to detect the collision between two object.

→ Explode : to simulate the explosion.

B. Diagram interfaces / heritages



C. Programming choice

1. Object with and without body

a) Abstract class

We chose to create abstract class to factor code and avoid some copy paste, it contains: (ObjectWithBody, ObjectNoBody).

We chose to split two object types, the one with a physics body and the ones without a body, the body use the implementations of the library jbox2D for the physics interactions.

b) Interface

We created two interface for the two objects types (ObjectCreate ,ObjectDraw).

c) Object

The objects who have physics interactions are: the garbage, the trashcan and the walls.
The bomb and the robot does not have a physic interaction, so they does not have a body:

→ for the robot as this one disappears, it does not interaction with anything else.

→ for the bombs we did not want the garbage or the bombs themselves to bounce on each other before the explosions.

That's why even if two bombs are next to each other and one explodes before, it will not move .
This can be improved in a future improvement.

2. A star

a) Path-finding

To move the robot, we must find a possible path from a starting position (the robot position itself) to a target position (the mouse position).
In this game, we are considering a position as a pixel on a screen
The path is a list of positions from the start to the end, each position follow next position to take.

b) Algorithm

→ Finding part:

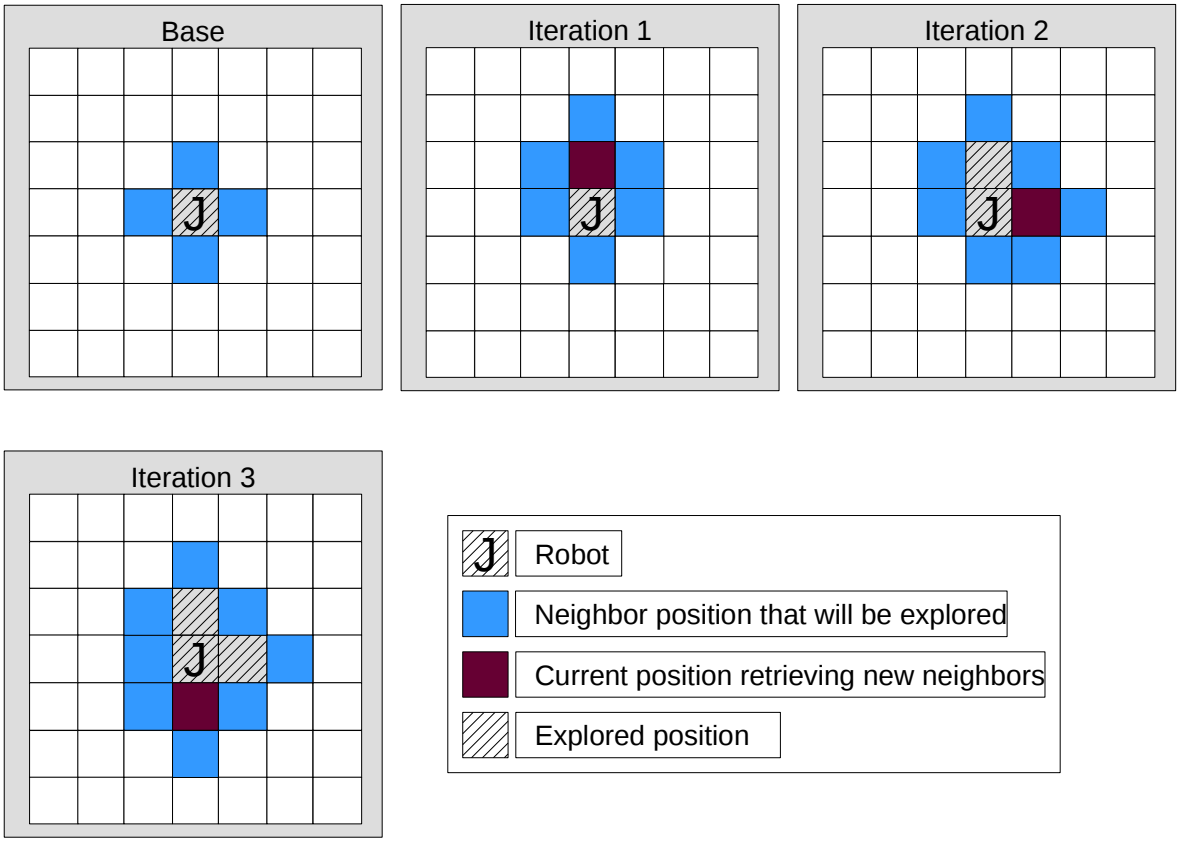
S, the start position
M, a map that positions who came from, it will be used to retrieve a path
Nothing came from S, so M contain the key-value S→null
Q, a queue, we put S in it
C is a position (current position)
While there are still positions to explore in Q (is not empty):
    C is the position retrieved from the head of Q,
    if C is the target position, break the loop
    for each neighbor positions of C (named N):
        if the position N has not been explored (meaning that N is not in M)
            M now contain the key-value N→C (the position N came from C)

→ Path building part:

If C is not the target position, return an empty array (it mean that no path is available).
At this point, a path has been computed inside M and C is the target position.
P is a List that will describe a path.
while C is not S (start position)
    put C in P
    C become the value of M[C]
put S in P
inverse P (we must inverse P because the computed path goes from the target position to start the position)
The algorithm end here.

BEN HAMOUDA Amel
HARDY Elise
RUBIO Julien

→ Iterations examples:



Using the A* version of the algorithm
The current path algorithm use the Breadth First Search algorithm, it mean that it will try every possible position to find the target position,
however, it is possible to optimize the algorithm speed by seeking in priority the positions who are the nearest from the target by putting a modification to the algorithm, named A*,
it will use with an heuristic value, calculated with the Manhattan distance to put a priority to each positions to be computed, the further a position is from the target, the lower is priority his (priority is set from 0 (highest priority) to +infinity (lowest priority)).
Also, the position now receive a moveCost, it mean that if the position is visited again (because of a wall blocking a path), the lowest one replace the cameFrom key, to to this position, because this position is reached faster than the previous one.

→ Priority positions implementation:

Some modifications are applied to the previous algorithm
A map will take a position as key and the priority as value
Note: because positions are never passed more than once, this implementation is compatible because the map will never have multiple same keys.
Each position visited have a priority value (Manhattan distance between the current position and the target position) the current position taken is now from the lowest priority.
If it never has been visited, it is added to the cameFrom map (with a movement cost), if it already exist in cameFrom (because of a new visit from another position) but is movement cost is lower, it is replaced with the current position, priority is the new movement cost + Manhattan distance.

→ Finding the lowest priority of the map:

The map is set as a list, then lowest value of the list is removed (highest priority).

→ Additional settings:

Some position might no be accessible, they can be skipped.
A position is not accessible if the robot may touch a solid object when placed at the position.

→ Additional optimization:

The target position is first checked to see if it is accessible before computing the path.

3. Specialized Class

In the project we have two specialized class. These class allow to use the library jbox2D.
The class Explode allow to simulate an explosion.
We use the interface RayCastCallback as well as the methods rayCast and reportFixture.
The class Collision allow to detect a contact between two fixture.
This class is used to make disappear the garbage when they touch the trashcan,

bounce each others when colliding themselves, and colliding the walls.

III. Improvements / Changes

We have brought improvements and changes since the beta presentation:

A. Improvements

→ Added the incrementation of the bombs when the player presses several times on the space bar.

→ Add the bomb explosions when the robot disappears (with the keyboard button B).

→ Move to the next level when the level is finished.

→ Restart the current level after 500 ticks if the level failed.

→ Optimization of the path-finding.

→ And if the player wants to leave the game before finishing all the levels just press Q (instead of pressing any button).

→ Added a build.xml file

B. Changes

→ We added comments and documentation.

→ We changed the HashMap in Map and the ArrayList in List when needed.

→ Exceptions were made when needed.

→ We put the names of the fields and methods in English.

→ We put the private, final when it is necessary.

→ We put a only one render frame.

→ We loaded the level when it is necessary, not all at the same time.

IV. Conclusion

The biggest difficulty of the project is the use of the graphical interface of Zen5 and object with body while it is not even seen in TP.
Finally, this project was more or less difficult because of the lack of time to carry it out, since there were the exams at the same time.

BEN HAMOUDA Amel
HARDY Elise
RUBIO Julien

V. Annex

A. Usage

To begin the game the user have to use the build.xml by using the command in the terminal:

→ ant compile
→ ant jar
→ java -jar wallj.jar

Then see the user guide.

B. Documentations

Inside the terminal:

→ ant javadoc

See the javadoc in the repertory docs/doc