

# Software Evolution Project

## Academic Year 2019-2020

### University of Mons

Tom Mens and Ahmed Zerouali

#### **Abstract**

This document presents the project assignment of the Software Evolution course. The project will be carried out in groups of two or three students and corresponds to 33% of the final grade of the course. The goal of the project is to apply in practice the concepts taught in this course during the theoretical sessions. The assignment requires each student to (1) understand an existing software system; (2) analyze and improve its quality; and (3) extend the system with new features.

## **Group Composition**

**GROUPE 1:** ROBIN WILLIEME, PIERRE ZIELINSKI

Selected extension: “Series of labyrinths”

**GROUPE 2:** XAVIER BOL, VIRGILE BROUILLARD

Selected extension: “Artificial Intelligence for PacMan”

**GROUPE 3:** REMY DECOCK , SAM BOSKOO, ROBIN SCHERER

Selected extension: “Special boxes and fruits”

# 1 Objectives

Each group will be assigned as many software systems as there are group members. The project will be carried out in five different steps (see below), each step corresponding to a specific objective of the project. The group will prepare **a single written report**, containing an analysis of the quality of the software systems and how their quality has evolved (i.e., improved or decreased) from Step 1 to Step 5.

Table 1 clarifies, for each step of the project, how students will be assigned to work on the different selected software systems.

Table 1: Steps to be followed by each group.

Group of two students	1: Quality Analysis Before	2: Quality improvement	3: Extension of systems by adding basic rules	4: Extension of systems with one new feature	5: Quality Analysis After
Student 1 Student 2	System 1 System 2	System 1 System 2	Group work	System 2 System 1	Group work

Group of three students	1: Quality Analysis Before	2: Quality improvement	3: Extension of systems by adding z rules	4: Extension of systems with one new feature	5: Quality Analysis After
Student 1 Student 2 Student 3	System 1 System 2 System 3	System 1 System 2 System 3	Group work	System 3 System 1 System 2	Group work

## Step 1: Understand and analyze the quality of an existing software system (individual work).

Each group member will select a different software system to start with Step 1. First, each student will become familiar with the structure and source code of the selected system, and with its functionality through executing the code. Then, the student will analyze the quality of the software system using a combination of static and dynamic software analysis tools.

**Step 2: Improve the software quality (individual work).** The goal is to improve the quality of one of the software systems by refactoring the source code; fixing bugs (if present); adding unit tests (if absent); and apply regression tests to validate the correctness of the program's execution. Each student will continue to work with the same system he worked with in the previous steps.

**Step 3: Extend each system by adding a set of basic functionalities (group work).** In this step, the students of each group will work together to implement a set of basic functionalities for all the selected systems. Depending on the system, some functionalities may already be present. The set of functionalities is presented in Section 5.

**Step 4: Extend a software system with new functionalities (individual work).** In this step, the students within each group will switch systems, i.e., the student that had been working with System 1 will

now work with System 2 and vice versa. Each student will add a new selected feature to one of the software systems (see Section 6 for the list of features).

**Step 5: Analyze the quality of the source code (group work).** In this step, the students of each group will work together to analyse the quality of the software systems after steps 3 and 4.

## 2 Technical Requirements

Throughout the project, students **must** use the distributed version control system Git, through a private repository hosted on GitHub, to facilitate the individual work as well as the group work.

All software systems must be compilable and executable with Java 8 or later, and JUnit (preferably JUnit 5) must be used as unit testing framework. The use of other additional testing libraries (e.g., mocking frameworks, and test coverage) is encouraged.

All software systems must be compilable and executable using **maven**, a build manager for Java projects. It should also be possible to execute the unit tests directly with maven. For more information, see <https://maven.apache.org>.

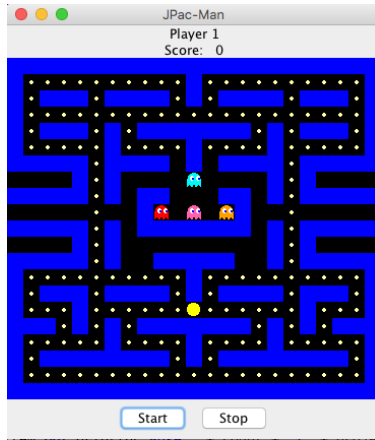
It is not required, but strongly encouraged, that students use a continuous integration tool (e.g. Travis, Jenkins, CruiseControl). Depending on the plugins installed, this allows to automate the test execution, coverage analysis and quality analysis, whenever code is committed to the version control system.

## 3 Detailed description of the project

This section describes in detail what needs to be done for the project. Each group and each student is free to choose the tools they will use for this project, as long as the choice is explicitly mentioned and justified in the **project report**. When choosing tools for your project, consider what you learned during the theoretical and practical sessions of the course, and the availability of software tools and libraries.

We have provided three different software systems that are similar in size, functionality and complexity. These systems implement the well-known Pacman game (details on the game rules are found in Section 5): Figure 1 presents the user interface of each Pacman implementation.

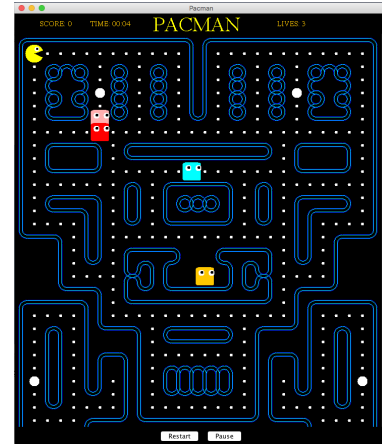
1. System 1: <https://github.com/ecos-umons/pacman-system1> (fork of <https://github.com/SERG-Delft/jpacman-framework>) is a very basic implementation of the Pacman game in Java, created by the team of Professor Arie van Deursen, Delft University of Technology (Netherlands). It contains several simplifications with respect to the Pacman gameplay. An incomplete set of unit tests is present in the source code.
2. System 2: <https://github.com/ecos-umons/pacman-system2> (fork of <https://github.com/philippwinter/pacman>).
3. System 3: <https://github.com/ecos-umons/pacman-system3> (fork of [https://github.com/rinus-mj/Pacman\\_INF-D](https://github.com/rinus-mj/Pacman_INF-D)).



System 1



System 2



System 3

Figure 1: Three Pacman implementations

Students must form groups of two to three members each. Each student must have a personal account and repository in GitHub<sup>1</sup>, and the repository of each student and each group needs to be shared with the teachers.

Each group will be assigned 2 or 3 of the 3 Pacman implementations by the teacher. The remainder of this section describes the steps each group will follow for this project.

A test-driven development process should be followed throughout the project. When refactoring the source code or implementing new features you must: (1) perform regression tests with the existing unit tests of the system in order to ensure that the initial behavior of the system has not been changed; (2) add new unit tests that are incomplete or absent from the existing test suite; (3) add new unit tests for the functionalities you will implement; (4) compute test coverage whenever you run the test suite. The test-driven development process involves steps 2, 3 and 4. You can use continuous integration tools to automate your testing activity and test coverage reporting.

<sup>1</sup><https://github.com>

### 3.1 Step 1: Analysis (individual work)

Each group member will choose which Pacman system they will start with. With his/her personal GitHub account, each member must copy to the group's GitHub repository the contents of the source code repository of the software system he will initially work with.

#### Get familiar with the source code

Each student needs to get familiar with the source code of the system assigned to him, in order to understand the structure of the source code and its associated unit tests, as well as to understand how the software works. At this point, students are free to use any code analysis tools to help them (e.g. tools presented during the theoretical or practical course sessions).

#### Software quality analysis

Each student will apply selected software quality analysis tools to analyze the source code and the unit tests of the software system he was assigned to. These tools will be used to find bugs, quality problems, bad smells, structural problems, etc. At the end of this step, each student will create a quality analysis report for the studied system. This report will be included as a section of the final group report to be submitted at the end of the project.

### 3.2 Step 2: Quality improvement (individual work)

From this step onward, each student will create a fork<sup>2</sup> of the private repository of their project and will use this fork to work on each task. At the end of each step, the student will use the pull request mechanism on GitHub<sup>3</sup> in order to integrate his activity to the central private repository of the project.

At this step, students will continue to work on the same software system as in Step 1. The student will use the quality analysis report he created for that system in order to start refactoring the source code of the system and improve its quality. Throughout the work he must perform regular individual commits and document the source code. At the end of this step, the student will document his refactoring activities, since they need to be included in a quality improvement report for the considered system, as a section of the final group report.

### 3.3 Step 3: Extend with basic functionality (group work)

In this step, all group members must collaborate to ensure that all systems contain the same basic Pacman functionalities. The description of the basic rules and functionalities is found in Section 5. If a system already implemented the required functionality, then no action is required. You will use your git repository to commit your source code and you must perform regular individual commits and document your source code.

---

<sup>2</sup><https://guides.github.com/activities/forking/>

<sup>3</sup><https://help.github.com/articles/using-pull-requests/>

During this activity, each student can either choose to directly commit to the central repository of your project, or to use forks and pull requests (as was the case in Step 2).

### 3.4 Step 4: Extend with new features (individual work)

At this step, students will switch projects, e.g., if a student had been working with System 1 in steps 1 and 2, he will work with System 2 in this step.

Each student must extend the systems with the functionality assigned to the group (as described in Section 6). This means that all group members will implement the same functionality, but each student will implement it in a different system using his own GitHub account and Git repository. Throughout the work the student must perform regular individual commits and document the source code properly. Therefore, the student's repository will contain all the changes made to the source code.

At the end of this step, each student will create a quality analysis report for the studied system after having added the new functionality. This report will be included as a section of the final group report to be submitted at the end of the project.

### 3.5 Step 5: Quality Evolution Analysis and Reporting (group work)

As the final deliverable of the project, the group must deliver a single quality analysis report on the entire group's activity, including in separate sections the reporting of the individual work of each student. The report must be submitted **in PDF format** on the course's Moodle platform before the due date. This report must contain a front page that clearly indicate the names of each group member, the academic year and the course title. The report must also clearly indicate which tasks were carried out by which member. The report should have the following structure:

**Section 1: Quality analysis of the initial versions.** This section will contain the quality analysis reports of step 1, for the initial versions of each analysed software system per group member (one subsection for each). A third subsection should compare the quality differences between the initial versions of the analysed systems.

**Section 2: Quality improvement.** This section will include a description of the refactoring and quality improvement activities of step 2, applied to each system per group member (one subsection for each). The description should include the improvements performed to: remove duplicated code, add tests, reduce quality warnings reported by the quality checker tools used during the initial quality analysis, improve the functionality of the code in case of errors or bugs, etc. Each subsection should explain the approach taken as well as a presentation and justification of the tools used.

**Section 3: Adding basic functionality.** This section will include a brief description of how the basic functionalities were implemented for each system, and include screenshots illustrating these functionalities. You must provide specific pointers to the Git version corresponding to the extended version of each system in order to assess the correctness of the implementation.

**Section 4: Adding new features.** Similarly to Section 3, each student must include a brief description of the implementation of the added features in the system assigned to him/her, and provide screenshots illustrating these features. Also, he must provide specific pointers to the Git version corresponding to the extended version of the system in order to assess the correctness of the implementation.

**Section 5: Quality Evolution Analysis.** This section should report an analysis of how the quality of the systems evolved during the different steps, and provide a comparison of the quality evolution between the considered systems.

**Section 6: Conclusion.** This final section should explain the difficulties or benefits you faced when working with each system with respect to the quality attributes of each system. Also, you can include other important observations, remarks or opinions related to the project and what you learned from applying software evolution tools in practice.

**Note:** In your report and source code you must explicitly mention if you have reused any external source code/libraries/etc., indicate the source and shortly explain the rationale for reusing each element.

**Note 2:** The history of the source code must be sufficiently detailed. This means that you must report the purpose of each version of the software and what is the difference of each version compared to the previous version. You can reference different commits on Git to facilitate this. Do not hesitate to tag certain commits to facilitate your work and that of the teachers.

## 4 Work distribution

The work will be done in groups of two students. You are free to form the groups, as long as each group consists of two students. If necessary, the teachers will randomly place students to each group to form the final groups or adjust the groups in case students did not find a group. Once the groups are formed, they cannot be modified and each group will be assigned with a unique identifier.

The final evaluation will take into account both the individual work, as well as the group work. Each student's activity in GitHub will allow us to analyze each student's work.

## 5 Pacman - Basic rules

*Pacman* is an arcade game created in 1980<sup>4</sup>. Many clones and variants of the game have been realized on various platforms, including PC. In the basic game, the player directs a yellow person in the shape of a camembert called **Pacman**. The goal of the game is to complete a series of levels. Each level consists of one labyrinth in which Pacman 🍷 walks. The labyrinth is also populated by **phantoms** which, for the most part, touch (and thus kill) Pacman. Pacman must avoid the ghosts and eat all the **gums** on his path. Also, he can eat **fruits** (🍎, 🍒, 🍊, 🍇, 🍓) to increase his score and gain extra lives. A level is completed as soon as Pacman has eaten all the gums in the labyrinth.

In each labyrinth there are four ghosts, each with one color and a unique name: 🐼 **Blinky** the red ghost; 🐼 **Pinky** the pink ghost; 🐼 **Inky** the blue ghost; 🐼 **Clyde** the orange ghost.

The behavior of ghosts is explained in Section 6.3. Each ghost tries to catch Pacman using a different method. Figure 2 shows a typical implementation of Pacman.



Figure 2: Pacman

[As part of the assignment the members of each group will work together to implement a set of basic functionalities for all the selected systems. These functionalities are presented below. Depending on the system, some functionalities may already be present.]

---

<sup>4</sup><https://en.wikipedia.org/wiki/Pac-Man>



## 5.1 Power pill

It is possible that Pacman becomes the hunter when he eats a **power pill**: the ghosts flee to avoid Pacman. If Pacman touches a ghost, it kills it. When a ghost is killed, it returns back to the starting pen where the ghost regenerates. The power pill has effect for a limited amount of time and after this time elapses, then all ghosts turn back to normal.

Modify the game so that it takes into account the presence of *power pills*. When Pacman eats a power pill, he gains 50 points.

For a short period of time, the rules of the game change (the game mode *escape*) and the prey becomes a hunter. When Pacman eats a power pill, then the ghosts are frightened. The timer of the level stops when he eats a power pill and the ghosts enter into flee mode: they become blue and their speed of movement is reduced to 50% (including Pinky). In flee mode, each ghost decides randomly which direction it will take. If Pacman touches a ghost during flee mode, the ghost disappears from the game and reappears in the middle of the labyrinth after 5 seconds.

If Pacman touches a ghost in leak mode, he eats it, which reports:

- 200 points for the first ghost he eats.
- 400 points for the second ghost he eats.
- 800 points for the third ghost he eats.
- 1600 points for the fourth ghost he eats.

There are four power pills per level. The first two Pacman swallows scares the ghosts for 7 seconds, while the two last for 5 seconds. After this period, the ghosts return in the mode in which they were before. The timer associated with this mode is picked up where it was stopped.

## 5.2 Pacman control - Continuous motion

In some implementations, Pacman moves one square each time the player presses a key. Normally, Pacman must continue to move forward as long as he does not face a wall and he does not receive any further instructions. If the instruction cannot be executed (for example, if the player presses the down arrow and there is a wall behind Pacman), then this instruction is simply ignored.

## 5.3 Labyrinths

At any level of the game, Pacman must be able to navigate in every labyrinth of the game. This means that any level must never contain a region/path that is disconnected from the paths that are available to Pacman. Figure 3 presents an example of two disconnected paths at the bottom left and right of the level. A disconnected region can become reachable to Pacman either by altering the walls to integrate the region to the labyrinths, or by adding passages which lead to the region. In the second case, the disconnected region must have one passage as entry point and a different passage as exit point.

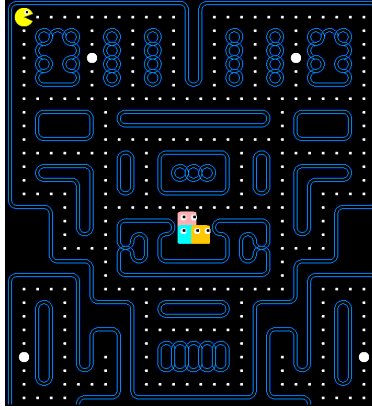


Figure 3: Example of disconnected paths

## 6 Additional Features for Extensions of Pac-Man Game

### 6.1 Series of labyrinths

Currently, the game ends when a ghost hits Pacman (and Pacman dies) or as soon as Pacman has swallowed all the gums in the labyrinth. Make Pacman at the beginning of a game of 3 lives. When touched by a ghost and if he has at least one life left, Pacman loses a life and is *teleported* on a random square which should be 4 squares away than the nearest ghost. When Pacman loses his last life, the game is over. Whenever Pacman earns 10,000 points, it gives him an extra life.

Also, make sure that when Pacman finishes a level with success, another level is put in place. Pacman starts this new level with the number of lives he had at the end of the previous level. Set up a system so that it makes it easy to indicate the order in which the levels are to be presented. Suggest a suite of some levels.

Each time a player has already achieved a certain level successfully, it will be registered by the application. The player can then decide before the start of each game, to choose the level he wants to start. (By default, this will be the first level.)

For each Pacman system, the levels must be represented in two different import formats, either as text (txt files) or as xml (tmx files). In order to generate valid new levels, you must also implement a level checker in the application. The level checker must confirm that all the board rules apply: a gum cannot be placed without a surrounding wall, each wall must be linked to another wall, etc.

Finally, you should implement campaigns for your Pacman application. A campaign is a series of levels and each level is played sequentially by the player. The next campaign only unlocks if all the levels of the previous campaign have been completed successfully. For each level a three-star success rating will apply: For each life Pacman loses in the game, a star is removed from this level but the starts can never be less than zero. If the player finishes all the levels of the campaign with 3 stars, then a bonus level is played.

**Note:** The developer of the Pacman\_INF-D system has also developed a system called Pcmn\_LevelEditor<sup>5</sup>. Figure 4 presents the graphical interface of this system. Optionally, you can include this in your implemen-

<sup>5</sup>[https://github.com/rinus-mj/Pacman\\_LevelEditor](https://github.com/rinus-mj/Pacman_LevelEditor)

tation and incorporate the level checker to this project.

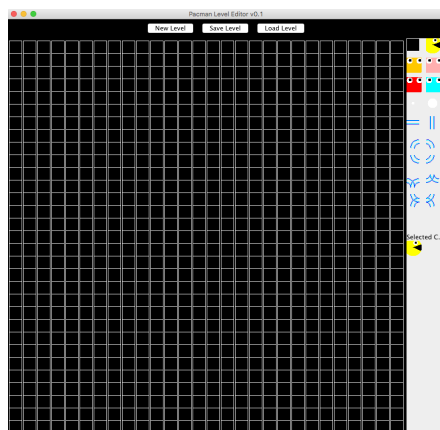


Figure 4: Level editor interface of Pacman\_INF-D

## 6.2 Artificial Intelligence for Pacman

Ensure that Pacman can be driven by Artificial Intelligence (AI) and not be controlled by a player. The goal is to obtain a final score as high as possible. You can see some strategies about how Pacman should move at the konket site<sup>6</sup>. Before starting a game, the player must be able to choose between controlling Pacman or being a passive spectator of the game. In the latter case, the AI must decide on the direction Pacman will follow only when Pacman is at an intersection. Once a direction is chosen, it is not possible to change Pacman's direction before the next intersection.

The behavior of Pacman must be implemented using the *Strategy Design Pattern*. This means that when Pacman is controlled by a player, is only one strategy. Your implementation must support a feasible and easy addition of new strategies: Each strategy must have all the necessary information in order to decide on the direction of Pacman. In no way should strategies be able to cheat by manipulating or giving directions which are contrary to the rules of the game.

## 6.3 Artificial Intelligence for the ghost behavior

Currently, the behavior of ghosts can be rather erratic, because the direction of each ghost is determined randomly. For each ghost, create AIs for the ghosts so they make more *interesting* decisions for the player. A decision is taken only when a ghost arrives at the position it wished to reach. The decision is based on the shortest distance between the current position and the position it wishes to reach. If several directions are similar, a ghost will always prefer to go up, then left, then down. A ghost can only go to the right if this direction is the only way to achieve the smallest distance to the destination.

The behavior of phantoms alternates between two modes: **pursuit** and **dispersion**. In **tracking** mode, each phantom has a specific and deterministic behavior and speed<sup>7</sup>:

<sup>6</sup>[http://archive.konknet.net/sngp.classicgaming.gamespy.com/games/pac/pac\\_faq.htm](http://archive.konknet.net/sngp.classicgaming.gamespy.com/games/pac/pac_faq.htm)

<sup>7</sup><http://gameinternals.com/post/2072558330/understanding-pac-man-ghost-behavior>

In **dispersion** mode, each phantom moves to its original position. These positions are located at the four corners of the labyrinth:

Once its original position is reached, the ghost will move in a circuit that consists always moving towards the left path for Pinky and Clyde, and the right for Blinky and Inky xee Figure 5)

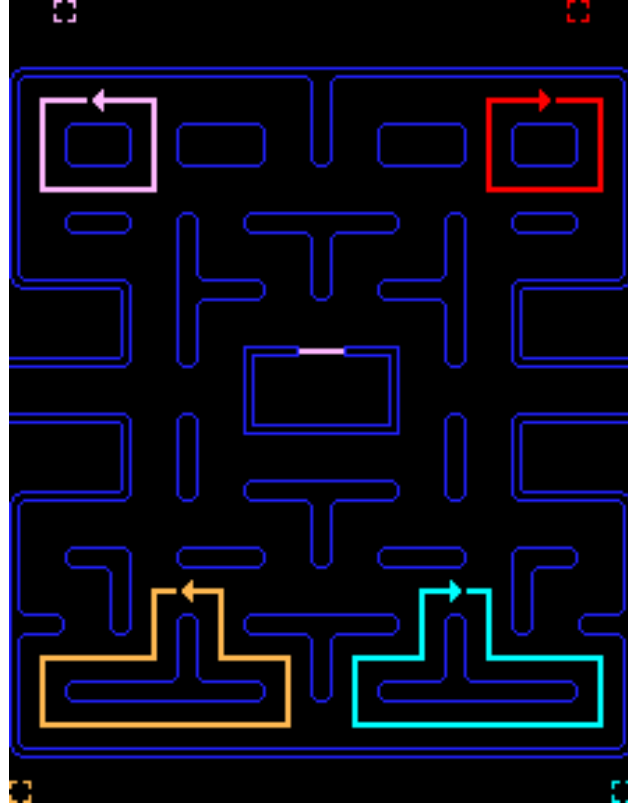


Figure 5: Ghost Paths for the dispersion mode

Changing between modes is defined as follows:

- Dispersing for 7 seconds, then continue for 20 seconds.
- Dispersing for 7 seconds, then continue for 20 seconds.
- Dispersing for 5 seconds, then continue for 20 seconds.
- Dispersing for 5 seconds, then continue indefinitely.

Use a *Strategy Design Pattern* to implement the behaviors of ghosts. The alternation between dispersion and pursuit will be achieved through a change of strategy. The common aspects of ghost behavior will be implemented in the form of strategies shared by the concerned opponents.

The system must support adding new strategies easily: strategies must have all the necessary information in order to determine the following moves. In no way strategies should be able to cheat by manipulating or giving directions which are contrary to the rules of the game.

## 6.4 Infinite Game / Map generation

In the basic game, Pacman must eat all the gums in order to move on to the next level. With an infinite game, the objective of the game changes. When Pacman moves up, a line at the bottom of the map disappears, and a new line is generated and added at the top of the map. Thus, Pacman can continue to rise (in principle) to infinity. As we move forward, the ghosts will move faster and faster, and there will also be more and more ghosts. The goal is to "survive" as long as possible escaping the ghosts while eating gums and fruits to maximize her score.

Note that in this version of the game, new ghosts may appear on a new line generated at the top of the map, the ghosts must therefore no longer be generated in the middle of the map.

## 6.5 Special boxes and fruits

### 6.5.1 Special boxes

This extension introduces special boxes on the game map. Some of these boxes can be predefined in the map, while others may appear and disappear randomly with a certain probability that depends on the level of the game or the score obtained.

The special boxes to be made will be:

**Traps** If Pacman or a ghost falls in a trap, it will remain locked on the space for a certain period of time.


**Teleportation** If Pacman arrives on a teleport box, he will be moved automatically and directly to another box of the game that is predetermined. Ghosts are not affected by teleportation boxes.


**Bridges** On some crossroads of the map there may be a bridge. It has the specificity that one can cross the hut without being killed even if there is a ghost under the bridge (or vice versa). Gums or fruits are always below the deck.


### 6.5.2 Special fruits and vegetables

This extension of the game introduces special fruits and vegetables that give Pacman a certain power or penalty if he eats them. This power will be instantaneous, or may have a certain duration. For all effects that have a certain duration, the application must clearly indicate this visually to the player. The availability of each special fruit or vegetable must depend on the level of play, the degree of difficulty, and the number of points obtained by the player for the current game.

**Positive effects (powers):**

**Grenade**  Eating this fruit has an instant effect of a *bomb*: all the ghosts that are within a maximum of 4 spaces of Pacman are killed instantly.

**Pepper**  If Pacman eats it, its speed *increases* for a certain period of time.

**Tomato**  If Pacman eats it, it becomes "invisible" for a certain period of time. This invisibility allows Pacman to cross ghosts without being killed.

**Red bean** 🍷 If Pacman eats it, it will launch a projectile in its current direction once per second for a certain period of time. A ghost on the path of a projectile will be killed instantaneously.

**Negative effects (penalties):**

**Potato** 🥔 If Pacman eats it, the speed of the ghosts *increases* for a certain predetermined period of time.

**Fish** 🐟 If Pacman eats it, he becomes "still" for a certain period of time.

## 6.6 Multiplayer mode with PacMan

This is a game of 2 players, where one player controls Pacman and the second player controls a ghost. Implement the ability to play the Pacman game for two people. The second player will choose which ghost he will control during the game. Pacman and the second player's ghost will use a *continuous shift* during their movement. For the first player (who controls Pacman) the objective of the game remains the same. For the second player (who controls the ghost), the goal will be to eat Pacman.

**Note:** The Pacman system supports multiplayer mode with Pacman but with two Pacmans. This feature indicates that there is only one Pacman in the game. The second player must control a ghost.

## 6.7 Multiplayer mode without Pacman

It is a game of 2 to 4 people, where Pacman does not exist anymore. Each player chooses as a character a ghost of a specific color (The remaining ghosts are controlled by the application). Each ghost has a predefined start position on the map, far away from the other phantoms. The goal of each player is to eat the most gums and fruits to maximize his score. The winner is the player who obtained the most points once all the gums are eaten.

If one plays with X people, the ghost of each player is a *hunter* for  $\frac{100}{X}\%$  of his time, and is *prey* the rest of the time. The game must clearly indicate at any time which ghost is hunter.

The prey type ghosts behave like Pacman, and can eat gums and fruit. The hunter-type ghost does not eat gums or fruit, but can hunt and eat other ghosts. When a prey ghost (i.e., a player) is killed by a hunter ghost, the hunter player gains points and the prey player loses points. A killed ghost is resurrected automatically to its starting position on the map after a predefined time.