ELECTRICAL AND INFORMATION ENGINEERING

University of the Witwatersrand, Johannesburg

Software Development II

Project 2020 — Super Pac-Man

1 Introduction

Super Pac-Man is an arcade game that was created in 1982 as a spin-off from the popular Pac-Man game. A screenshot from the game is given in Figure 1.

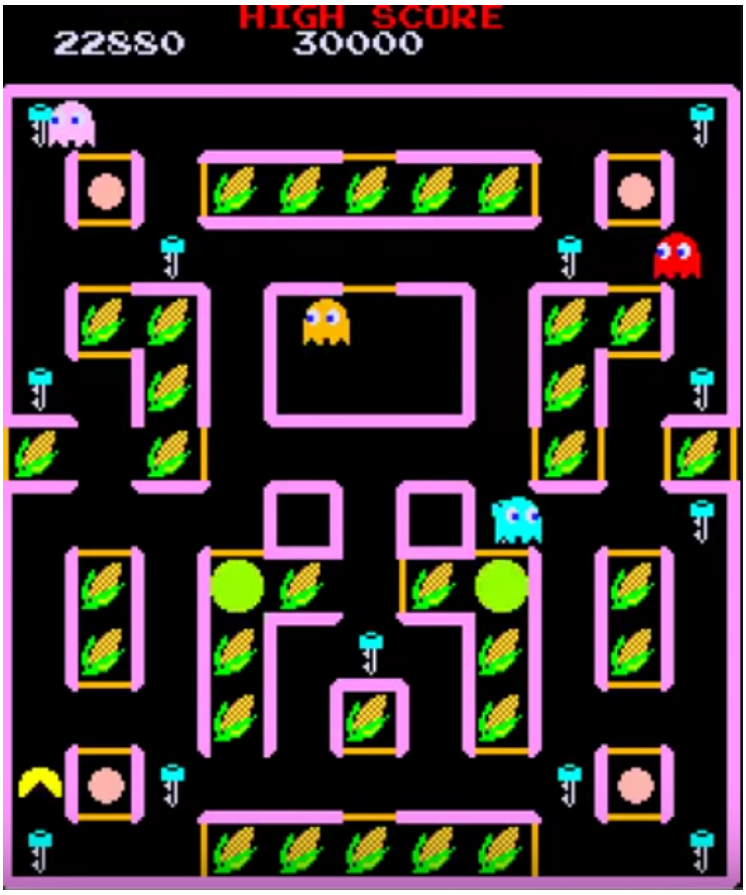

Figure 1: Screenshot from Super Pac-Man

Wikipedia [1] describes the game as follows:

> "Sound and gameplay mechanics were altered radically from the first two entries into the Pac-Man series—instead of eating dots, the player is required to eat keys in order to open doors, which open up sections of the maze that contain what in earlier games were known as "fruits" (foods such as apples and bananas, or other prizes such as Galaxian flagships), which are now the basic items that must be cleared. Once all the food is eaten, the player advances to the next level, in which the food is worth more points. In earlier levels, keys

> unlock nearby doors, while as the player progresses through the levels, it is more common for keys to open faraway doors. Pac-Man can enter the ghost house at any time without a key.
>
> In addition to the original power pellets which allow Pac-Man to eat the ghosts, two "Super" pellets are available and will turn Pac-Man into Super Pac-Man for a short time. In this form, he becomes much larger, can move with increased speed when the "Super Speed" button is held down and has the ability to eat through doors without unlocking them. He is also invulnerable to the ghosts, who appear thin and flat in order to give the illusion of Super Pac-Man "flying" over them. He still cannot eat them without the help of the original power-up. When Super Pac-Man is about to revert to regular Pac-Man, he flashes white. The Super power can then be prolonged by eating a power pellet or super pellet, if available."

The task in this project is to write a keyboard-driven version of Super Pac-Man. This implies that your game is primarily based on the same game mechanics as Super Pac-Man (see Section 4) but you are free, *and encouraged*, to choose any game theme and backstory that you like.

2 Project Outcomes

On completion of this project you should be able to:

- Perform an object-oriented decomposition/analysis of a software problem which involves a variety of interacting objects.
- Design and implement an object-oriented solution in C++.
- Understand and use existing software libraries in conjunction with your own code.
- Provide a test suite verifying the correctness of your software.
- Provide the requisite documentation for a software project, including an automatically generated technical reference manual.
- Work successfully in small team to deliver a software product.

3 Constraints

The game needs to be coded in ANSI/ISO C++ using the *SFML 2.5.1* library [2]. Earlier versions of SFML may not be used. The game must run on the Windows platform.

The emphasis of this project is on *good object-oriented design* and not on fancy graphics. Good graphics are only regarded as a minor feature enhancement. With regard to the graphics, these restrictions apply:

1. The dimensions of your game window (when maximised) must not exceed 1600 $\times$ 900 pixels.
2. You may not use OpenGL.

You may not use any other libraries or frameworks that are built on top of SFML.

4 Game Mechanics

The game mechanics have been described above but it is advisable to watch the video at https://www.youtube.com/watch?v=KYpuKc86gT8 so that you can see the original game being played. A good section to watch is from 1:50 to 3:34. This section illustrates all of the mechanics.

5 Categorisation of Features

The following section describes the basic functionality that you are required to implement for your game. Additionally, features which are considered minor and major enhancements are listed. If these are implemented well, higher marks will be awarded in the *Functionality* category (see the assessment grid and Table 1). If you want to implement a feature that is not listed, please contact me first so that I can advise you. You are welcome to include audio in your game; however, this is not regarded as a feature.

5.1 Basic Functionality

The following is considered basic functionality:

- The following game objects exist: the maze containing fruit in (initially) locked sections, Pac-Man, a couple of ghosts, and keys to open doors to the locked sections of the maze.
- Pac-Man is controlled by the player and can move around the maze but not through the maze walls or locked maze sections.
- The ghosts move on their own throughout the maze, and any opened sections, and they chase Pac-Man. They cannot move through the maze walls or into unopened sections. If any of them touches Pac-Man, he dies, and the game is over. The manner in which they chase Pac-Man can be simple, implying that, at times, the ghosts might be stuck in certain parts of the maze when trying to reach him. The ghosts at the start of the game are located in the centre of the maze.
- Whenever Pac-Man moves over a key, it disappears, and one, or more, doors to the locked sections of the maze open allowing Pac-Man to enter and eat the fruit. If Pac-Man eats all the fruit contained in the locked sections, the game ends with Pac-Man winning.

5.2 Minor Feature Enhancements

Minor enhancements include, but are not limited to, the following:

- The maze contains at least two *power pellets*. When Pac-Man eats one of these the ghosts change colour and can be eaten by Pac-Man. This state lasts for a short period of time. If a ghost is eaten it re-spawns in the centre of the maze. *Implementing this feature is mandatory if you are aiming for Good or Excellent for functionality.*
- Graphics are good (not composed of simple shapes such as rectangles and triangles).
- There is some sort of scoring system and display. High scores are saved from one game to the next, and can be viewed.

- Stars appear periodically in the maze and can be eaten for extra points. This implies that the above feature must be partially implemented as well.
- The player has more than one life and the remaining lives are depicted on the screen.
- The structure of the maze is read in from a file.

5.3 Major Feature Enhancements

Major enhancements include, but are not limited to, the following:

- The maze contains at least two *super pellets*. When Pac-Man eats one of these he enlarges and can now pass through any of the locked doors, opening them in the process. His movement speeds up. The ghosts appear flattened and Pac-Man can pass over them, neither eating them nor being killed by them. This state lasts for a short period of time. If, in this state, Pac-Man eats a power pellet then he can eat the ghosts as described in Section 5.2. *Implementing this feature is mandatory if you are aiming for Excellent for functionality.*
- The ghosts intelligently chase Pac-Man without ever becoming stuck in the maze, moving clearly in the wrong direction, or moving aimlessly.
- The game includes an in-game maze editor allowing the player to create, edit, and use their own mazes. These mazes are saved in files.

6 Project Submissions Overview

Agile methodologies are a popular method of developing high quality software. They are both *iterative* and *incremental* in nature. Iterative implies that multiple passes through the phases of the software development lifecycle take place. On completion of each iteration a working build is produced that has a subset of the total functionality that is required. This is usually released to the customer in order to elicit early feedback. Each iteration results in an increment in functionality and iterations continue until the final solution is achieved. Note that your code will probably change significantly between releases. This is to be expected as your design converges on a final solution.

In order to encourage this style of development, there will be two interim submissions and a final submission. The deadlines are available on the course website. The first two submissions are submissions of work-in-progress and a reduced set of project deliverables is required. The final hand-in will require the submission of all of the project's deliverables. The deliverables required for each submission are summarised in Table 2 and described in more detail in Sections 6.1 and 6.2.

The School's Late Submission Penalty Policy will be strictly applied to the final deadline. The policy must be read and understood by the student. A late penalty will also be in effect for the first and second submissions. Refer to Section 7.3 for more information on this.

6.1 First and Second Submission Deliverables

For the first submission you are expected to show exploratory use of the SFML and doctest libraries. Very simple game and test functionality is expected. For example, having only a single game object which is capable of being moved around, and some associated tests, is sufficient. For the second submission you are required to build on the functionality of

Functionality present in second submission	Highest rating achievable for *Functionality*
Less than basic functionality	Acceptable
Basic functionality	Good
Basic functionality plus one major feature	Excellent

Table 1: The highest possible rating that can achieved for the *Functionality* category is determined by the functionality present in the second submission.

the first submission. Each submission or release is required to have a splash screen which *correctly* informs the user about the keys used for playing the game.

To ensure that you leave yourself sufficient time towards the end of the project to complete any remaining functionality, testing, and the all-important documentation, the functionality that is achieved by the second submission deadline affects the highest rating achievable for the *Functionality* category (refer to Table 1). Functionality will only count as being achieved if it has been decently implemented.

In order to promote the writing of tests concurrently with the writing of code there is a requirement with regards to the amount of testing that has been achieved by the second submission. Specifically, *basic movement tests must be provided for all of the game objects* that have been implemented by this time. A five percentage-point penalty will apply if this is not achieved.

The first two submissions do not require any reports. Only the release notes, source code, test code and the corresponding (working) executables need to be submitted (refer to Table 2). These submissions must meet the requirements given in Section 7. Non-compliant submissions will be penalised (see Section 7.3).

Deliverable	First & Second Submission	Final Submission
Game executable, associated files, and source code	✓	✓
Splash screen with playing instructions (part of exe above)	✓	✓
Test executable(s) and source code	✓	✓
Published release notes on GitHub	✓	✓
Project report		✓
Declaration		✓
Technical reference manual		✓

Table 2: Deliverables required for each submission

6.2 Final Submission Deliverables

Each project group (a group of two) must submit the deliverables listed in Table 2 and comply with the requirements in Section 7. In addition to the deliverables required in the earlier submissions, the final submission must include:

- A *Project Report* (no page limit) presenting the problem, its analysis and specification; the conceptual model of the domain; an overview of how the code is structured; the various classes/class hierarchies and their responsibilities; the dynamic behaviour of

key parts of the system; and a critique of both the final functionality achieved and the object-oriented design. Additionally, it must include a short section explaining which game functionality has not been tested and why. *You are required to provide an abstract for this report*.

- A *Declaration* form digitally signed by both project partners stating which partner did which parts of the project and specifying each partner's discretionary mark which will be added to the project mark (see Section 8.2). If no discretionary marks are specified, or if the form is not signed by both partners, then the marks will be forfeited.
- A low-level *Technical Reference Manual* explaining the source code to other programmers who might wish to understand and modify it. This manual must be automatically generated using Doxygen.

7 Submission via GitHub

7.1 The Project Repo

The project repository, which will be made available to you, must contain the following items located in the correct directories:

1. All the *source code* for the game in a directory called `game-source-code`.
2. All of the *test source code* in a directory called `test-source-code`.

Do not include in your repo:

- The source code for the SFML library or the doctest library.
- Files that are produced as a by-product of the build but are not necessary for running the game. Such files include project files produced by the IDE, object code files, and so on.
- Any executables, associated libraries, or game resources.
- Any documentation files.

7.2 Project Releases on GitHub

There are three aspects to a published project release on GitHub:

1. A tagged commit in the project repo forming the release,
2. A release title and release notes, and
3. A zip file containing the executables, library files, game resources, and documentation.

These aspects described in more detail below. Refer to the Git/GitHub guide for instructions on how to create a release using Git and GitHub.

7.2.1 Release Tags

A commit on the *master* branch needs to be tagged for the release. The tags for the releases are to be named *exactly* as follows: `v1.0` (for the first submission), `v2.0` (for the second submission) and `v3.0` (for the final submission).

You can try out this process by creating a test release before the first submission is due. For example, you could publish a test release with the tag `v0.1`.

7.2.2 Release Notes

Release notes need to be published on GitHub for each release describing the functionality available in the release.

7.2.3 Executables, Game Assets, and Documentation

Upload a *zip file* containing the following directories to the *Assets* section of the GitHub release:

1. A *single game executable* and a *single test executable*, along with any required *DLLs* and any other files that are needed for the game and tests to run, in a directory called `executables`.
2. A *PDF* of the project report in a directory called `docs` (only required for the final submission).
3. A *PDF* of the declaration in the `docs` directory (only required for the final submission).
4. A Doxygen-generated `HTML version` of the technical reference manual in the `docs` directory (only required for the final submission).

7.3 Penalties for Non-Compliance

Five percentage points will be deducted from the final project mark for each submission not meeting the submission requirements. This is independent of the evaluation of the deliverables submitted. A submission is non-compliant in the following circumstances:

- The release is not correctly published on GitHub (Section 7.2).
- The release is missing one or more of the required deliverables. For the final release, any missing deliverable will result in a rating of *Unacceptable* on the rows of the marking grid where the deliverable is assessed.
- The executables cannot be run for any reason. Note, the executables will be run directly and not run from within an IDE. You may not assume that the MinGW compiler is installed on the computer running the game. Refer to the Git/GitHub guide for the libraries that need to be submitted along with your executable.
- There are less than *five commits per group member*, prior to the release (and post the previous release, for the second release onwards) on `master`. These commits can be made directly to `master` or can be merged into `master`. Only commits which are linked to a group member's GitHub profile are counted.
- The project repo and/or the assets zip file do not have the required directory structure.
- The assets file is not a zip file but instead is some other archive format.
- The game's source code is duplicated within the `test-source-code` directory.
- The project constraints are violated (see Section 3).
- For the second submission only: basic movement tests are not provided for each game object that has been implemented.

Five percentage points will be deducted from the final mark for each interim submission that is submitted late. If the second submission is not received by 16:30 on the day of the deadline then, in addition to a five percentage point penalty, it will be taken that no

submission was attempted and the highest achievable rating for the *Functionality* category will be *Acceptable*.

8 Assessment

8.1 The Assessment Form

The assessment form, which forms part of this project brief, indicates how the project will be assessed. Each category is rated from *Unacceptable* to *Excellent*. Each of these ratings corresponds to a particular mark (shown below). The overall mark is determined by averaging the category marks.

Rating	Mark
Unacceptable	0
Poor	20
Acceptable	55
Good	70
Excellent	95

Table 3: Ratings and associated marks

If any category receives a score of *Unacceptable* then both students' overall marks are capped at 40% and both students' discretionary marks (see below) are forfeited. Note, however, that the overall mark can be lower than 40%.

8.2 Teamwork and Group Self-Assessment

The ability to work well with others is fundamental to engineering, and to software engineering, in particular. This is why teamwork is explicitly one of the outcomes of this project. You may not work on your own and project partner changes will only be considered when a student de-registers or there is clear evidence that one member of the group is not contributing. You are required to act professionally and support each other in achieving the goals of the project. It is a good idea to state your expectations from each other before the project begins and to discuss how you will handle possible conflicts that could arise.

Both group members are expected to contribute fairly equally to the project. Each group of two is allocated ten percentage points in discretionary marks. It is up to the group to determine how to divide this. The division must be in terms of *whole numbers*; decimal numbers will be truncated. The discretionary marks may be evenly split if it is felt that both members contributed equally to the project. If this is not the case, the group member who has made a greater contribution can be acknowledged by granting them a larger share of the marks. The discretionary percentage points for each group member are added to the overall mark to determine that member's final project mark. In order for the discretionary marks to be granted both group members have to agree on how the percentage points are apportioned and *sign* to this effect on a declaration form (refer to Section 6.2).

In order to account for gross differences in contributions to the project, the following rule applies: If the source code additions (see the *Insights*|*Contributors* page on GitHub) of one member of the group is less than a third of the other member then the overall mark of the member having the smaller contribution will be capped at 40%.

8.3 Early Hand-In Bonus

Groups who submit *all* of their project deliverables by 17:00 on the day before the final deadline, will enjoy a bonus of five percentage points, and a relaxed evening.

9 Plagiarism

Plagiarism detection software will be used to compare project submissions to one another and to code available on the internet. All instances of plagiarism will be severely dealt with. No two groups may have identical or overly similar deliverables.

10 Final Thoughts

It is critical that you apply a balanced effort to all aspects of the project and that you do not fall into the common trap of over-focusing on coding, and functionality, and neglecting other important activities such as design, critical analysis, testing and documentation. Remember that implementing additional functionality will in turn require more effort in all other areas of the project. It is invariably better to produce a final product that is acceptable in all respects, rather than a product that is excellent in only one respect and poor in all others.

Good luck and have fun!

References

[1] Wikipedia. "Super Pac-Man." https://en.wikipedia.org/wiki/Super_Pac-Man, Last accessed: 21/08/2020.

[2] L. Gomila. "SFML." http://www.sfml-dev.org/index.php, Last accessed: 21/08/2020.

University of the Witwatersrand, Johannesburg
School of Electrical and Information Engineering
Software Development II

Discretionary Mark: **5**

PROJECT ASSESSMENT FORM V0.36

Highlight one block on each row of the marking grid
by right-clicking with the mouse

Discretionary Mark: **5**

Bonus + Penalty Total: **0**

Bonus, penalty and discretionary marks have *already been applied* to the marks shown.

	Unacceptable	Poor	Acceptable	Good	Excellent
Problem Understanding, Solution and Evaluation: project report	extremely flawed problem understanding/specification/ conceptual solution, key functionality and/or design choices not explained, presentation of solution does not match implementation	poor problem understanding/ specification/conceptual solution, key functionality and/or design choices hardly explained, class responsibilities inadequately described, blind acceptance of clearly defective functionality	adequate problem understanding/specification and conceptual model, class responsibilities described, some description of dynamic behaviour, minimal/flawed critique of the final solution in terms of functionality and design	good understanding/specification and conceptual model, class responsibilities well described, reasonable description of dynamic behaviour, reasonable critique of the final solution in terms of both functionality and design	astute understanding, specification and conceptual model, class responsibilities and dynamic behaviour are well described, excellent critique of the final solution in terms of both functionality and design, consideration of the broader problem domain
C++ Design and Implementation: source code	SFML 2.5.1 not used, implementation violates constraints, not object-oriented: no user-defined classes, GitHub not used for version control	poorly chosen abstractions or many key abstractions missing, poorly designed class interfaces, inappropriate relationships between classes, patent violation of fundamental principles such as DRY, implementation more like C than C++	abstractions generally have acceptable/appropriate behaviour but some key ones may be missing, acceptable class interface design but implementation may not be well hidden, mostly acceptable class relationships, modern, idiomatic C++17 mostly used	*2 out of 4* : 1) well-modelled abstractions at all levels of granularity with good interfaces which hide information 2) clean separation of presentation and logic layers 3) small classes and no long functions 4) good use of role modelling. No clearly wrong design decisions, modern, idiomatic C++17 used	*3 out of 4* : 1) well-modelled abstractions at all levels of granularity with good interfaces which hide information 2) clean separation of presentation and logic layers 3) small, cohesive classes and no long functions 4) good use of role modelling. No clearly wrong design decisions, modern, idiomatic C++17 used
Functionality: game executable	no executable, executable does not run, executable not built from submitted source code	application has major functional flaws, no splashscreen with playing instructions	all basic functionality working acceptably, splashscreen with playing instructions	all basic functionality working plus 3 minor features (including power pellets) OR 1 major feature and 1 minor feature (power pellets), splashscreen with playing instructions	all basic functionality working plus 2 major features (including super pellets) and 2 minor features (including power pellets), splashscreen with playing instructions
Automated Testing: test executable and source code, test section in project report	no genuine attempt at unit testing, doctest framework not used	insufficient testing - not meeting the requirement for *Acceptable*	test coverage of game logic is adequate and includes basic movement and collision testing for *all* game objects, some important game logic is not tested, adequate test section in report	test coverage of game logic includes all classes/functions responsible for the movement and collision of game objects, either directly *or indirectly,* testing is thorough and test code is of good quality, good test section in report	distinguished from *Good* by one or more factors: comprehensive coverage of all game logic, advanced use of testing framework, use of a mocking framework, automated tests given for difficult-to-test functionality eg. involving randomness, gui interactions etc.
Technical Communication: project report and technical reference manual	report deviates significantly from the School's standards, technical reference manual not generated using Doxygen	report does not conform to the school's standards, use of language, style and tone is poor, report structure is poor, poor technical reference manual	report mostly conforms to the school's standards, use of language, style and tone is acceptable, report structure is acceptable, acceptable technical reference manual	report mostly conforms to the school's standards, use of language, style and tone is good, good use of diagrams to communicate concepts, report is well-structured, **acceptable abstract**, good technical reference manual	report fully conforms to the school's standards, use of language, style and tone is excellent, good use of diagrams to communicate concepts, report is well-structured, **good abstract** and technical reference manual
Comments:					

Notes:
All categories are equally weighted
If any category receives a rating of Unacceptable then both students' marks are capped at 40%

Bonus and Penalties:
Non-compliant submissions: first: -5; second: -5; final: -5
Early hand-in: +5; Late final submission: within first hour: -5; before 16h30: -15