2805ICT: System and Software Design
3815ICT Software Engineering
7805ICT Principles of Software Engineering

Overall Project Description
(INDIVIDUAL SUBMISSION)

COMPLETION OF THIS PROJECT CONSTITUTES
Milestone 3

Vladimir Estivill-Castro
Larry Wen

July 6, 2020

1 Complex Computing Characteristics

The student who completes this item of assessment satisfactorily will demonstrate competency in the following *Complex Computing Characteristics*.

1. Involves wide-ranging or conflicting technical, computing, and other issues

   because it involves the design of software architecture, the analysis of requirements, consideration of sophisticated advanced new versions, and deployment to several platforms.

2. Has no obvious solution, and requires conceptual thinking and innovative analysis to formulate suitable abstract models

   because the tasks puts into practice sophisticated modelling techniques, software building tools, version controls and software documentation generators.

3. A solution requires the use of in-depth computing or domain knowledge and an analytical approach that is based on well-founded principles

because the tasks brings together sophisticated methods for software development and the practical illustration of software engineering principles.

4. Involves infrequently-encountered issues

   because students face complex architectural design issues: the project offers challenging algorithmic tasks and user-interface issues as a complex generalization of a classic game are requested as part of the deliverables of the project.

5. Involves diverse groups of stakeholders with widely varying needs

   because the requirements are significantly open and milestone 2 demands the production of a progress report to meet the needs of diverse stakeholders.

6. Has significant consequences in a range of contexts

   because there are issues in the design of the GUI when buttons are not rectangles, or where graphs must be embedded in the canvas, but also in terms of splitting software components to create re-use opportunities in client/server architectures, for instance.

7. Is a high-level problem possibly including many component parts or sub-problem

   because there are issues of gamification and accuracy of the game versus the complexities of an interactive application on several platforms.

Objectives

1. Explain design principles including of least privilege and fail-safe defaults, separation of concerns, information hiding, coupling and cohesion, and encapsulation.
2. Describe the design process for a software development project for each of the main software design methods
3. Create appropriate system models for the structure and behaviour of software products from their requirements specifications
4. Use a design paradigm to design a simple software system, and explain how system design principles have been applied in this design.
5. Select an appropriate software architecture as the design basis for a given software requirements specification, justify the selection based on its advantages over alternative architectures.
6. Create software programs that make use of appropriate design patterns.
7. Create user interface software using either event driven or callback based designs
8. Explain the importance of Model-View controller to interface programming.

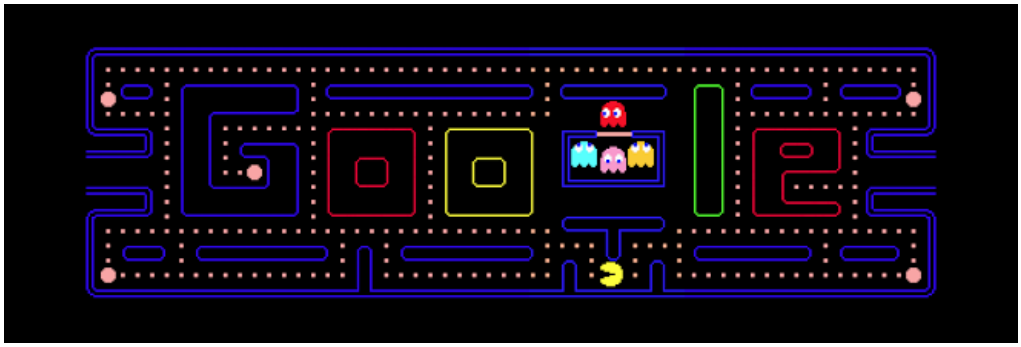

Figure 1: Snapshot of the Pac Man as illustrated by Google for its 30th anniversary.

9. Discuss the properties of good software design including the nature and the role of associated documentation.

10. Create appropriate design documentation for a variety of different designs.

Background

The literature on games acknowledges that qualities of a good computer game are a continuous challenge, engaging storyline, flexibility, immediate as well as useful rewards and combining fun and realism. There are some games that have become iconic landmarks of classes of games. Take, for example, Tetris that in 2014 became 30 years old and is considered the world's most successful game. However, in this project we focus on instances of another famous computer game: the well-known computer game Pac Man.

The Pac Man [1] game was celebrated by Google when it turned 30 years old

(www.google.com/doodles/30th-anniversary-of-pac-man),

and this year, Pac Man turned 40!

1.1 Playing the Game

The Wikipedia page for *Pac Man* [2] has some information about the way one plays the game.

Pac Man is set in a regular rectangular grid arena where a chase happens. The player drives a characteristic icon (the *Pac Man*) in the rectangular maze that has corridors. The player can only move to adjacent cells North, East, West or South if there is no wall or the boundary of the arena. The foal is to eat all of the fruit placed in the arena while avoiding four coloured ghosts. They ghosts are identified by their different colours and their names are Blinky (the red one), Pinky (the pink one), Inky (the cyan one), and Clyde (the orange). The ghost chase the player. We will start with a very simple level, where each ghosts plays one move random and one move in the direction of the players token (as if there were no walls); that is we assume the ghosts have a compass that provides them the direction of the player. If a player collects all fruit in the maze, When

[1] There is apparently an official website: www.pacman.com/en

[2] en.wikipedia.org/wiki/Pac-Man

all of the dots are eaten, the player advances to the next level. If Pac-Man makes contact with a ghost, he will lose a life; the game ends when all lives are lost. Each of the four ghosts have their own unique, distinct artificial intelligence (A.I.), or "personalities". Wikipedia claims that Blinky directly follows the *Pac Man*, Pinky and Inky try close corridors and strategicly block escape routes of the *Pac Man*, and Clyde will swap behaviours, sometimes giving chase and other times fleeing from the *Pac Man*. For the first implementation, the artificial intelligence of the ghost is just to have less frequency of a random move with respect to a move in the direction of the *Pac Man*. A video that demonstrates how the game works is www.youtube.com/watch?v=i_OjztdQ8iw.

Task 1

Your first task is to implement in your favourite programming language and using your favourite programming tools a version of Pac Man where scores are proportional to the time taken to collect all fruit for a particular dimension of the board. However, this is just one phase of the overall challenge, and you should consider designing your software architecture and software components so other tasks are also achieved.

Instructions

However, many variants of the game are possible. Your job is to design software components and software architectures that enable significant re-utilization as well as other properties of software quality. We are interested in your infrastructure supporting other version of the game as well as potentially many platforms (hardware and software).

More importantly, you should complete this overall project including a report that discussed

1. your design decisions,

2. and why your decision making is following software engineering principles.

Thus, for example, many versions of Pac Man are discussed in the re-makes and sequels of the Wikipedia page.

In particular, for this project we suggest the implementation of several variants.

1.2 The presentation layer of the game

We suggest that the maze should be able to change to a grid of hexagons. In this geometric maze, the characters (the ghosts and the *Pac Man*) should be able to move to any of the 6 adjacent cells to the current cell unless there is a divider or the cell is at a boundary. This type of grid is illustrated in Figure 2 for another game.

An additional (third type) environment would be completely topological, where the maze is simply a graph and the characters travel along the edges of the graph, while occupying nodes.

Task 2

Your second task is to implement in your favourite programming language and using your favourite programming tools a version of Pac Man where scores are the time taken for a particular dimension of the board. However, this time your program should enable three version of the game, (1) an array of square cells, (2) an hexagonal grid and (3) and arbitrary graph.

Figure 2: Snapshot of the *Mine sweeper* GUI as a grid of hexagons.

Other game extensions

The next point your program should illustrate is *sophistication of the* strategy of the ghosts. You should implement Dijkstra's shortest path algorithm and have at least one monster capable of making a move along the shortest path (again, the levels may introduce a degree of fidelity to the shortest path, so that in lower levels there is more randomness, while in higher levels, the ghost follows the path with seldom random deviations). Since Dijkstra's algorithm may be costly consider implementing a version of $A^\star$ and argue for an admissible heuristic.

In the original version of *Pac Man* there was a fixed number of levels and the maze were predefined (loaded from a file). You should add an option to have the software to generate the arena (its walls) dynamically, maybe with some randomness, and even the position of the fruit. So the player does not necessarily learn the maze but has to face a slightly new arena every time, even at the same level. We name this feature a *dynamic maze generation*.

The other game extensions should be provided with *options for the user*. For instance, the size of the grid for the maze (the arena, ie, the environment) should be able to be selected by the user. Or perhaps how much it grows from one level to the next (from a minimum value to a maximum value). The user may opt for an option by which the user can consume an energy booster that last for some time (or within a zone). While on the booster status, the *Pac Man* can kill the ghost and the ghost will not reappear for some time and only from the home of the ghosts. The user should be able to regulate how much the monsters become faster from one level to the next.

Task 3

Your third task is to extend your implementation so that game extensions regarding the strategy of the monsters is sophisticated, the mazes are generated dynamically, and parameters of the game can be selected by the user.

Tools

You are free to chose the programming language and if you prefer, you can also use a GUI designer. You should use version control software and also a framework for Test-Driven Development. Optionally you can use a tools for Continuous Integration. All of these are tools of your own choice.

Constraints

You must ensure that the final version of your project is capable of running in at least 2 very different platforms (two operating systems). One of them must be a recent version of Linux and using a GUI. Alternatively, the second platform could be a Web-enabled architecture or a mobile device (tablet, smart-phone) on a different operating systems (Windows, MacOS). Alternatively, you can produce the 3 versions to run over a client/server architecture that hosts several clients and keep scores ranking several competitors.

You must ensure that significant software components are common and have been factorized in both deployments, ensuring maximum re-usability. You may argue the same for several design artefacts.

First Challenge.- Implementation and Iterative development

Consider the project as a series of challenges. Plan for each and everyone of them. It is crucial that you progress from one version of your software to the next but with a plan to eventually achieve all suggested requirements.

If you do not complete all requirements, you must at least show that the approach you took follows software engineering principles.

As has been indicated before, the first challenge of this project is to create software programs that make use of appropriate design patterns. In particular, the three variants proposed by Task 1, Task 2, and Task 3 are implemented preserving a robust software architecture that maximizes software reuse and software quality.

Second Challenge - Reflective illustration

The second part of this project is to provide design documentation that demonstrated the use of software design principles. In particular, you should direct the marker evaluating your submission with your documentation to specific examples on how your project

1. applies principles like separation of concerns, information hiding, reduction of coupling, maximizing cohesion and applying encapsulation.

2. uses a design method or explain how your product evolved by design into the shape and form that it is now

3. explain how the structure of your code follows a Model-View-Controller software pattern and justify this as variations of the Model, the View or the Controller could potentially be introduced

4. has been lifted to design documents for communication its design (you must include at least one class diagram, two collaboration diagrams, one sequence diagram and one state chart)

5. has anticipated the potential for faults and failures, and what testing is performed to validate components.

What shall you submit or demonstrate

A demonstration video

You should submit the link to a video (no more than 20 minutes long) whose first part is to demonstrate the running computer game, but then later part should expand significantly into the description of the software architecture and the design. The video should also show the compilation phases, or project infrastructure of using an IDE. It should also demonstrate the use of test-driven-development and the outcomes of introducing variants in the code.

Your log of your version control software

You should submit the log of your version control repository in compressed from but should be easily readable in ASCII text once decompressed.

Final reflective report

You should submit a *Reflective Challenge Report* in PDF format which should go over the examples in the video as well as the artefacts of your design and guide the marker as suggested above. It should be easy for the marker to find merit in your submission and to find that your submission meets the objectives in the marking criteria. You should demonstrate the objectives set at the front of this project description.

Marking Criteria

This assignment is allocated 10 marks as follows.

Fully functional implementation of Task 1	1 mark
Fully functional implementation of Task 2	1 mark
Fully functional implementation of Task 3	1 mark
Implementation of Task 1, Task 2, and Task 3 is cross platform	1 mark
Evidence of good software development practices (version control, unit testing)	1 mark
Report and video proficiently illustrate Objective 1	0.5 mark
Report and video proficiently illustrate Objective 2	0.5 mark
Report and video proficiently illustrate Objective 3	0.5 mark
Report and video proficiently illustrate Objective 4	0.5 mark
Report and video proficiently illustrate Objective 5	0.5 mark
Report and video proficiently illustrate Objective 6	0.5 mark
Report and video proficiently illustrate Objective 7	0.5 mark
Report and video proficiently illustrate Objective 8	0.5 mark
Report and video proficiently illustrate Objective 9	0.5 mark
Report and video proficiently illustrate Objective 10	0.5 mark