Task 1: Major Project Proposal

Defining the Problem

- *Possible client identification*

Possible clients for this software would include anyone who enjoys playing strategy games. The clients would have to be aged around 10 years and up to ensure that they are able to understand the game and play through each stage successfully.

- *General description of the situation and the context of the problem.*

The current market has larger amounts of FPS or RPG games than anything else. This leads to a lack of games in other genres, such as strategy or card games. Most PC games are competitive and online multiplayer, which contributes to a lack of free offline games in this genre. A software for a single player, turn based card game would be an appropriate solution for this problem.

- *Client's needs from a solution*

Functionality requirements

- The entire program will run on a single window
- The system will detect keyboard and mouse inputs through which the user will interact with the game
- A max framerate of 60 fps will be allowed
- The user will be able to open a menu interface in game, allowing them to exit or restart the game

Compatibility requirements

- The system will be designed to run on a computer
- It will run on all operating systems
- The program will take keyboard and mouse input and use numerical and boolean values to process information.
- Input:
    - Mouse
    - Keyboard (optional)
- Output:
    - Screen
    - Speakers (optional)
- No internet access required as it is an offline single player game

Performance requirements

- The system will need to be able to respond fast enough that there is no significant delay between user inputs and display updates
- The file size should be small and would not require a large storage space.
- As the system is offline, the performance will be entirely dependent on the device's processing speed. It will not need to rely on fast internet connection etc.
- Algorithms will be as optimised and compact as possible
- Fps from 30 - 60. Game will run at lower frame rates, but graphics will not be as great.

- *Design specifications of the software system.*

Concept

The proposed system will be a turn-based card game, where the user plays against the rules of the game itself. The game will feature endless levels and features to upgrade the character card or modify strategies. The system will be offline and single player.

Gameplay

The user will control a character card and navigate that card through a 5x5 grid of other cards to reach an exit. Each turn the user will be able to make a set number of moves, and the character card will interact with other cards that are adjacent to it or on the character card's movement path, depending on their individual features.

Once the player reaches the exit, the cards on screen will be refreshed signifying the next level/stage. Every 10 levels, the background of the game changes. Stage 100 has a unique background. This process will repeat until the character card loses all its health points. The user will be able to use coins obtained during the game to purchase upgrades from an in-game shop.

Components of User Interface

- Main menu
  - Pause/exit menu (variation of main menu)
- Game screen
  - Features 5x5 grid of cards
- Shop screen
- Extra: (These features are not necessary for the game, but may be added if there is time after development of the main system)
  - Leader boards screen
  - Collectible items screen

User interaction

- User starts the game on a main menu, that will have buttons which link to the other screens
- When the character card is clicked it will be selected
    - When adjacent cards are clicked, or arrow keys are used, the card will move, after confirming that this movement is valid according to game rules
- When other cards are clicked, an information tab will appear
- When the pause menu tab or shop tab is clicked, the respective screen will be displayed
- Current Items will be displayed on the shop tab, but basic stats will be visible on the main game screen
- Sound output volume will most likely be controlled by the user's own device

Code Structures

- The code will be built using:
    - Functions for main operations
    - Classes for similar objects
    - Loops for repetitions
- This will minimise manual repetitions of code and decrease the length of the program. This will improve the speed at which the program will run, and storage space required

Language

- Running the program on Python allows it to be compatible with all devices, after the initial setup.
- The initial setup involves installation of libraries such as PyGame.
- The system will attempt to automatically install but this may not always work due to differences in syntaxes across operating systems or user permissions

Ergonomics

- The menu screens can be accessed with shortcut keys. E.g., press 'p' for the pause menu.
- Cards will be large enough to be easily clicked on by the user.
- Once the character card is clicked it will be able to be moved by arrow key controls instead of purely mouse clicks
- WASD will also work instead of arrow keys
- Basic aesthetics will be used to ensure the game looks appealing and will be improved upon if there is enough time.

System Specifications

Minimum specifications to run the game:
(The game may still run if these specifications are not met)

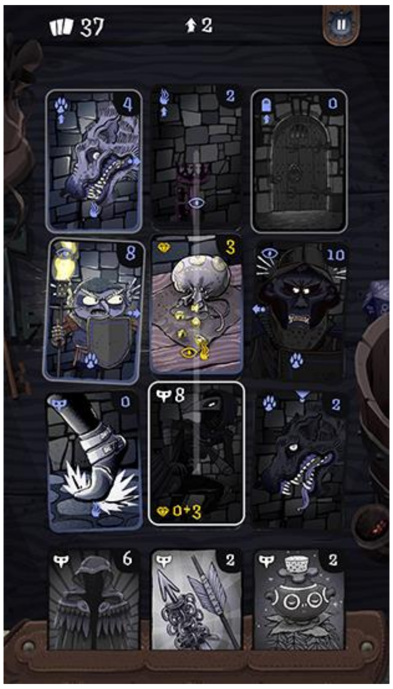
- Compatible with Linux, MacOS and all versions of Windows OS after Windows 7
    - The program only automatically installs PyGame for Mac and Windows OS
- Requires a Python interpreter or IDE
- Requires Python 3.10
- Input: Mouse
- Output: Screen (The game will scale to accommodate all screen sizes)
- RAM: 2GB
- CPU: Intel® Core™ i3 or equivalent
- HDD/SSD: 1GB
- Graphics card: Any graphics card with video memory capacity 500MB or greater

Due to the game's small size and simplicity, only a small amount of RAM is required. This also means that a weaker CPU would be able to run the program. The system employs basic PyGame graphics, which means that a more powerful graphics card will not be required. The system being able to run on all operating systems allows it to be suitable for the broad range of possible clients. This is also supported by the system only requiring basic input and output devices, such as a mouse and screen, both of which are already built into most laptops.

As the game is to be developed with Python, the device will require a large storage space and a Python IDE to be installed. Python 3.10 will be used to take advantage of new algorithm structures such as case selections, to optimise the code.

The system will only need to handle one user per device as it does not employ a network, which significantly improves the performance.

References to Real World Systems

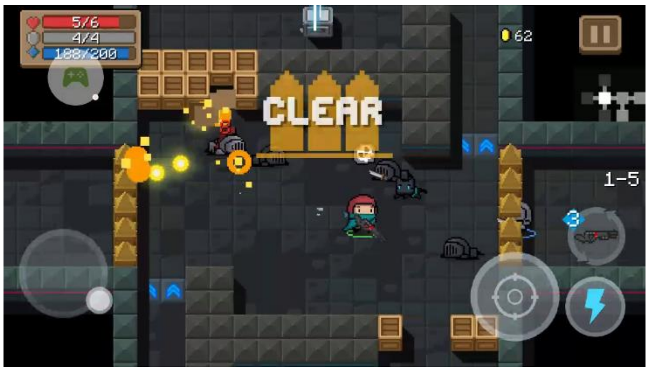


Card thief
This game is the primary source of inspiration for the proposed system. The game revolves around a character card navigating through a 3x3 grid of cards while stealing as much gold as possible.

The movement of the character card through the grid will be implemented within my own system, but without diagonal movement. Just like this game, as the character card moves, it will interact with any card in its path, and after the card has moved, adjacent enemy cards will attempt to move towards the character.

This game also features different types of cards, such as items and coins, different enemies and empty doors and hallways, which will be used for my own project.

The appearance of the cards in my own game will also be partially inspired by the cards in this one.



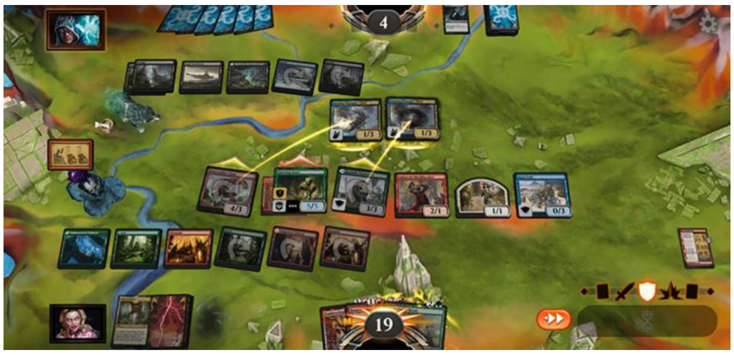
Soul knight
Soul knight is a game where the player navigates a character through multiple stages, while defeating enemies on the way. The character becomes more powerful by equipping better weapons found in-game.

The concept of gaining different items in game will feature in my own system and will be unique for each playthrough. Like Soul Knight, my game will also feature items of different rarities and features.

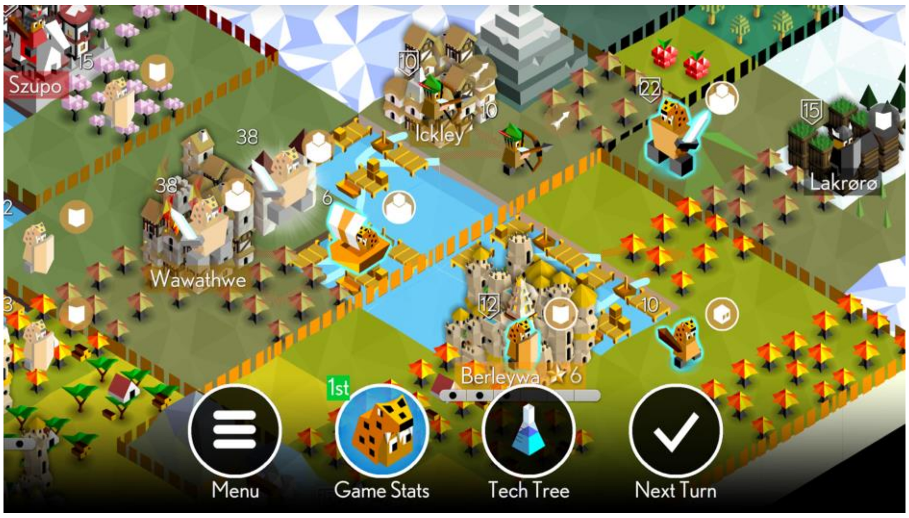
While each playthrough usually ends when the player reaches level 3.6, it also contains an endless mode, where the stages in the game loop after that end stage. This inspired the concept of endless levels in my own game, where the game levels randomly generate after floor 100.

MTG Arena
The main inspiration taken from this game is the way card statistics and effects are managed. Each card has a number for damage dealt and health, as well as individual icons that represent each special ability or effect the card has. This system will be adapted for my own game, but extra features such as movement will also be implemented.

The aesthetics of these cards will also be used as inspiration for my own game.



Polytopia
The characters in this game move through a large grid, and this grid movement will be implemented within my own game. Another important aspect of this game are the various types of terrain distributed through the grid. Each tile that does not hold a city can be a mountain, forest, ocean or empty and each form of terrain provides different effects. This concept will be used within my project, to increase the variety of cards in the game.



Software Development Approach

The software development approach that has been chosen to develop this major project is evolutionary prototyping. Prototyping is an iterative approach, where a model of one aspect or the entire system of the final product is created, used when the full requirements for a product are not known. Each successive prototype is checked in accordance with the original requirements by the client and developers and the process is repeated with a refined prototype. With evolutionary prototyping, the original model created will 'evolve' into the final product through continual testing and improvements, unlike other types of prototyping approaches that discard models after testing and receiving client feedback.

Benefits of evolutionary prototyping that facilitate the development of this system include its low cost and short time. Since the project, being a school assignment, is a small-scale project, it has no budget and a short time frame of a few months, which matches the features of prototyping.

The flexibility of evolutionary prototyping is another feature useful for this project. As there is currently no clearly defined vision or detailed requirements for the final product, the constant testing and redefining of the system specifications, featured in this approach will greatly help with its production. This would allow the development of this system to adapt and implement changing requirements and refine previously defined requirements through these tests. This repetitive testing will also ensure that each aspect of the product will perform its required function and reduce the amount of testing required once the product has been completed.

The simplicity of prototyping, compared to other approaches such as agile or structured, is another factor that contributes to its suitability for this project. The lack of structure means that less time is dedicated to implementing a solid plan for the final product, which would enhance the efficiency of the development process. The agile and structured approaches are also much more expensive to use and require a well constructed plan, resulting in their unsuitability for this project.

Usually, the prototyping method would involve an increase in interactions with the client, increasing the time taken for development of the system, due to a need to schedule meetings and present the prototype in a way the client can understand. However, this system is being developed for a school assignment and has no real client, so its development will not be affected by this issue.

While primarily implementing this system using evolutionary prototyping, elements of Rapid Application Development (RAD) will also be used. This includes the use of existing systems, such as Python modules and libraries as well as recycled functions from other projects.

Programming Language Selection

The programming language that will be used to create this system is Python. This language possesses multiple features that would help address the specifications of the proposed system.

Python consists of extensive modules and libraries developed by the community, which would help with aspects of the game such as detecting keyboard and mouse input and displaying a user interface. This saves substantial amounts of time that would otherwise be spent manually programming these features. These modules are reliable, as Python is an open-source language, and most common modules are constantly checked by members of the community.

Familiarity with the language and its algorithm structures would significantly improve efficiency of developing the system and would also help with learning how to use new features and modules. This optimization is further improved by the simple syntax of Python, allowing the code to be shorter and written faster. Python is compatible with all operating systems and is an interpreted language, facilitating the frequent testing performed with the prototyping approach, as the entire code does not need to be checked and translated to machine code, before running the program. Programming tools or IDEs for Python, such as Repl.it and PyCharm also exist, which assist with the testing process and can identify syntax errors before running the code, further optimising the process.

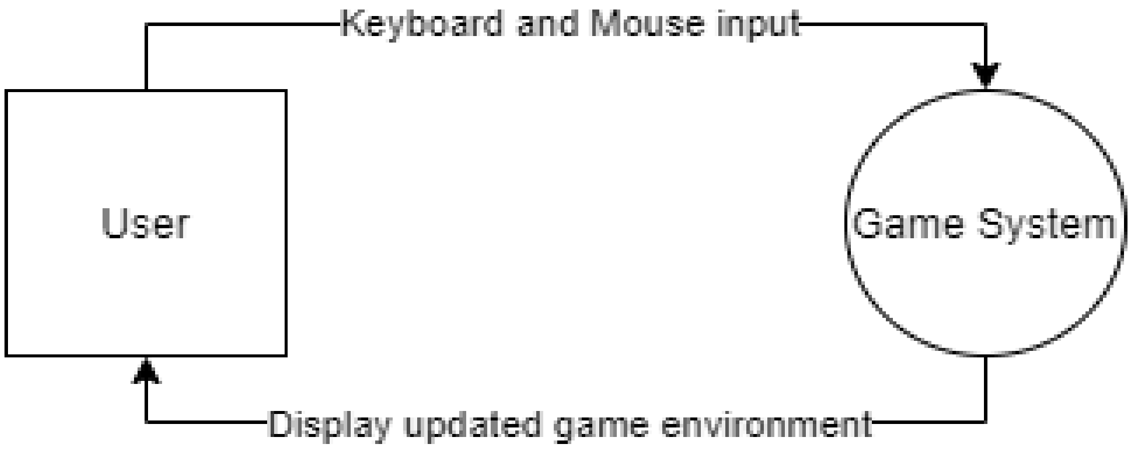
Some disadvantages of using Python as a programming language for this system involve its high memory usage due to its flexibility in data types, but this should not significantly affect the program as the program itself should be very small. However, python, being an interpreter language, means that the program will run slower and consist of runtime errors that would require extra testing to eradicate.

System Documentation

- *IPO Diagram*

Input	Process	Output
Keyboard Input (arrow keys OR WASD)	Determine character path	
	Validate character path	
	Move Character	Display character movement
	Calculate interaction with nearby cards (attack enemy, collect item card)	Display interaction
Mouse Input	Identify clicked location co-ords	
	Identify button or card clicked	
	Execute button process, or get information on card clicked	Display button process (e.g., Open menu or shop) or card information on updated screen
	If character card is clicked: (Alternate way to move character) ● Check for other nearby cards clicked ● Get character movement ● Validate movement ● Move character ● Calculate character interaction	Display character movement and interaction

- *Context Diagram*



Quality Assurance

Efficiency
Efficiency will be ensured by reducing repetition of code and decreasing its size through classes, functions, and loops. There is also no need to connect to a network and the only system that will run in parallel with the game will be sounds, which can be disabled.

Integrity
There is no need to check input data, as the program will automatically ignore all invalid input. This is because the program will only search for mouse clicks in valid areas at a given time. Only specific keyboard input will work at a time. The program does not handle real world data, so it does not need to verify or update any.

Reliability
Reliability of the program should have no issues as the code does not rely on real world data or external data. No important data is saved on files, etc and the game will always run if the system requirements are met.

Useability
The general user interface is very simple, as most interactions between the user and the system are point and click. Shortcuts such as letter keys to open menu/shop and arrow key movement for the character are available. A tutorial explaining the basic game controls, accessible from the menu, will also be included.

Accuracy
Code will be thoroughly tested due to the nature of the prototyping approach. Other users will be involved with final tests to ensure nothing is overlooked. Comments will be used to ensure that any function can be easily modified, without having to relearn what a section of code means.

Maintainability
Comments throughout code will improve maintainability and test data will be provided in documentation. As few 'hard-coded' values will be used as possible. Almost everything will be stored in at least a variable, so that it can easily be changed later if required.

Testability
To improve testability, other potential users and other students will be involved in final tests and test data will be provided with documentation. Code divided into separate functions will allow the entire sections of code to be tested individually, speeding up the testing process.

Reusability
Factors such as independent functions and classes will enhance the reusability of the system. Comments throughout the code, and whitespace to divide the code into sections, will also help with reusability.