# CIS 434 Software Engineering

# ***Python Python Game***

## Group 12

Aidan Zapotechne | Derek Woods | Toral Zaveri

4 / 20 / 20

# Team Contributions

This section highlights our contributions, due to the small size of our team we were able to work together on most features of the project. A detailed breakdown of these features can be found in the repository or ‘README.md’.

**Aidan Zapotechne:**
Project Proposal
Professional Awareness, Gantt Chart, Conclusion

Software
- Settings menu (buttons and changing game settings based on button clicks)
-2-P snake collision handling
-Fruit spawns based on amount specified
-2-P race timer
-Obstacle implementation

Final Report
- Abstract
- Project Description
- Professional Awareness

**Derek Woods:**
Software:

- Base Game (V1.0.0)
    - Simple framework of a single snake with movement, growth and collision detection to kickoff the project
- Game Settings
    - Not to be confused with the settings menu, game settings was a python class instantiated as the gs object. gs was used to collect, update and pass all game variables, settings and objects between all classes. This allowed for cleaner code and a much easier time for the developers to update settings and pass information.
- Various bug fixes and UI improvements

Final Report:

- Game manual in ‘README.md’
- Partial Objectives
- Project Timelines

**Toral Zaveri:**

Project Proposal:
Abstract and Objectives

Software:

- Adding live score for classic game
- Adding live score for 2-player game

Final Report:
Conclusion

# Abstract

This project aims to recreate the fun and simplicity of the classic and well-known Snake Game. Our software implementation was written in Python, and mainly utilizes the Pygame python library to bring the game to life. No other libraries were used, other than the shelve library to store the high score locally on the user's machine. This project explores a new dimension to the traditional snake game by adding a game options menu featuring new game modes, and settings to customize the experience. We believe this has made our game more interesting, enjoyable and challenging.

The steps to completing this project involved building the board for the game, using the pygame library to implement drawing game objects, finalizing the movement and game collisions, and creating different game modes by modifying the existing settings. Additional features addressed in the code include maintaining up to two different snake objects, calculating snake movement along all segments of the snake, maintaining game setting in a respective class, and menu navigation. The simplicity of this game makes it an ideal candidate for a new implementation to add extra functionality, gamemodes, and settings. This outcome of this project has hopefully changed the way people think of traditional snake games.

The process utilized for creating this project credits much of its success to the material of our class, Software Engineering. By following proper guidelines including following a software process model and establishing engineering requirements, our team was able to work efficiently and effectively. Another aspect of this was source control and documentation. Through the use of github, we were able to track all progress through release notes, and efficiently share code. The use of git's branching tools allowed us to maintain a master branch, which was always functional, and also maintain test branches which tested new features before merging them into the master branch. All of these techniques put together led us to our finalized project.

# Objectives

The objectives of our project did not change from the ones we originally set out. Our original objectives gave us plenty of aspects and features to focus on and we were able to achieve them all. The only exception would be a cooperative 2 players mode, which we swapped out for a second competitive 2 player mode which was more enjoyable for the players.

- Utilize the PyGame framework in Python to create an iteration of the snake game.
- Have all the basic snake game aspects like snake growth, collisions and arrow key controls.
- Allow the difficulty to change during the game.
- Make sure the game has an enjoyable user experience.
- UI improvements & better sprites
- 2-player mode, with either competitive or cooperative game design
- Different game modes, including multiple food spawns, borderless mode, etc.

# Project Description

## Project creation process

The full process for creating PythonPythonGame consisted of the following. To start, each team member setup their github account to link to our master branch. We verified understanding of git controls and each cloned the repository to our local machines. Next we delegated game features to each team member. Member's would select based on the Gantt chart features that needed to be implemented and try their hand at them. Team meetings would sometimes take place after class to serve as a way for member's to go over what they have accomplished so far, and seek help if needed. Between team meetings and source control via git, we also maintained proper documentation with each major release of the game. By incrementally adding all features to the project, addressing bugs as needed, and maintaining proper documentation, we finally arrived at our final release, and with it our final report.

## Software features

PythonPythonGame supports several ways for the user to customize their experience. To start, the user is shown a main menu in which they can start the game with default settings, or open the settings menu. In the settings menu the user can change five different attributes about the game. The first, and most significant setting is the gamemode. The standard gamemode is one-player, and is aptly named classic mode because it is the base game. The second is two-player race, a mode in which two players must compete to either eat more fruit than their opponent in a given time, or see their opponent hit an obstacle before time is up. The third and final gamemode is called melee, and it features a spin on the previous two-player mode. Instead of collecting fruit within a time limit, there is no fruit and both players incrementally get larger overtime. The winner is the last one standing, or rather the player who

does not collide with anything. The second option specifies the board size, allowing the player to select a range of five different boards that vary the amount of rows and columns are traversable in game. The third option allows the player to specify how many fruits are spawned on screen at one time. The fourth option allows the player to specify if borders should either be obstacles, or pass through borders where the player appears at the opposite end of the board when passing through a wall. The fifth and final option allows the user to specify obstacles spawning, in which five grey tiles will spawn randomly on screen that the player must avoid.

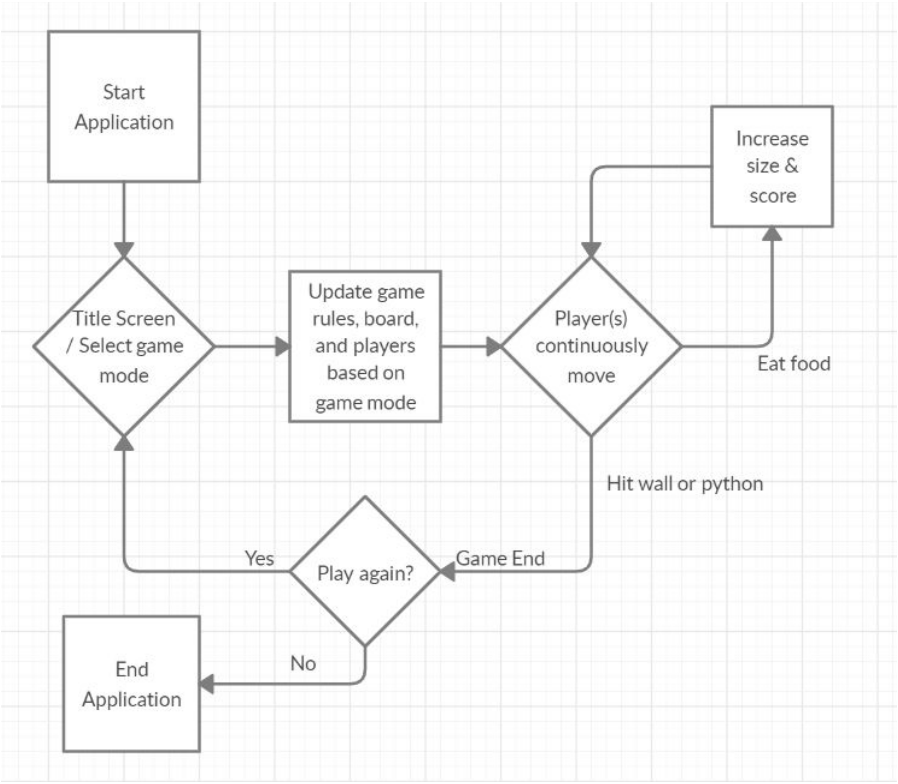
## Project difficulties and solutions

Video game software presents many challenges the development team must address. There are so many variables that all must update real time, often very quickly to simulate movement and interaction in-game. Some of the significant examples our team faced included:

- How to implement snake movement, specifically the snake's segments (long body) ?
    - This was solved by using python's built in data structures in a clever way. By using the enumerate function, we can store the snake's body positions in a list. Then, when movement is determined, all segments update by following a seperate list which also contains all turns that take place. The full implementation can be seen in the source code.
- How to implement different game modes?
    - This was solved by creating various menu buttons in pygame. Two lists were also maintained, one including which buttons were active and one including which were not active. By toggling different buttons to and from each list, we would then update game settings based on which buttons were active. The full implementation can be seen in the source code.
- How to maintain the high score?
    - The high score is saved to the user's local machine through the python shelve library. The score is saved to a small scale locale database so to speak, and is retrieved each time the game boots back up to preserve the value.

## Charts & Graphs

Below is our UML state diagram. It is relatively straightforward, thanks to the simplicity of the snake game.




Improvements given more time

Given more time, more game modes and settings would be a logical way to build upon the game. Since there are already several ways to customize the game, even more would continue to make it a more robust game. Other than additional settings, the game could also benefit largely from a graphical update. Proper sprites for the snake and better UI for the active playing field would be a great improvement.

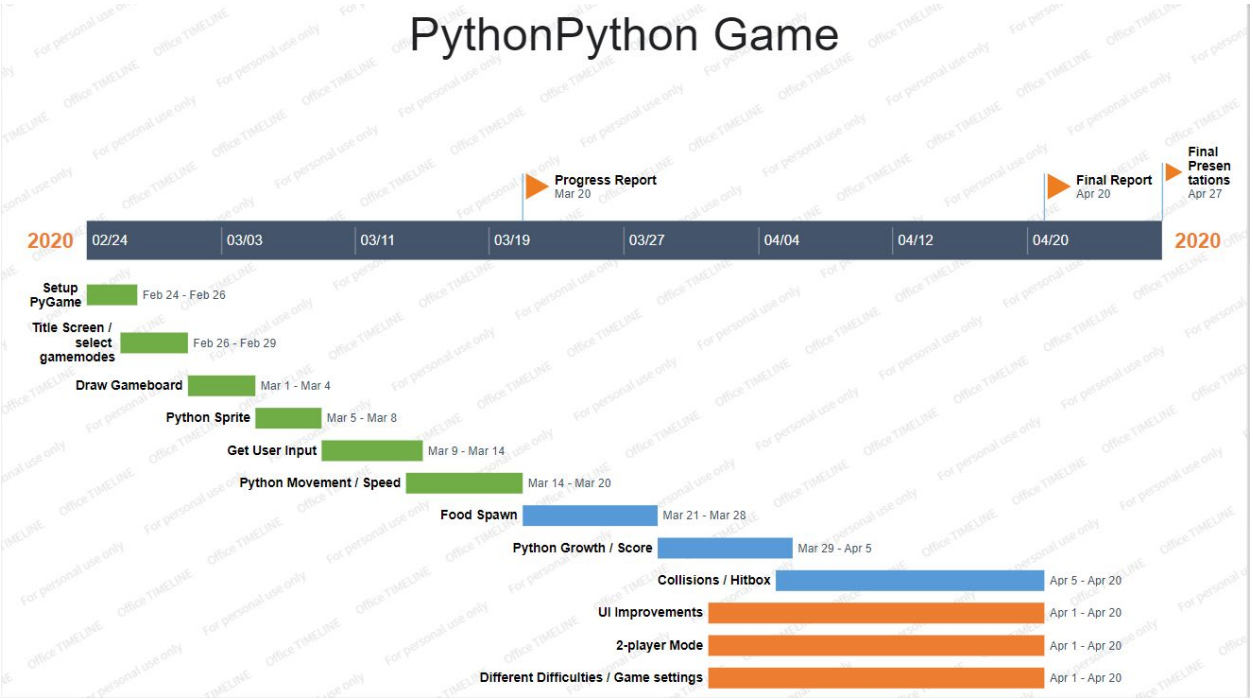
## Professional Awareness

In order to be professionally aware whilst creating our version of Snake game, it is important to understand the copyright and trademark laws at play. Snake game was created by the company Nokia, meaning we must respect any ownership they have over their creation. Copyright in itself would apply to the source code of Snake, in which we have nothing to worry about. Our snake game will be coded from the ground up solely by us, meaning we would break no copyright laws against Nokia. Trademark is another issue, in which Nokia owns the trademark of “Snake game”, the name itself. This means to complete our project while also respecting Nokia’s trademark, we must come up with our own name for our project.

# Project Timeline

## Gantt chart

Below is the gantt chart for a project. Thanks to prior planning, research and knowledge of what our project would require we were able to follow our gantt chart almost perfectly. The exception was our second task of the title screen and game modes.




## Tasks

The team meticulously documented all versions, features and bugs of the project during development and this can be found in README.md. This will be a quick summary of that documentation in a list of tasks and a mention of how it relates to the gantt chart.

1. PyGame setup and draw game board were the first tasks completed.
2. Python sprite, user input, movement and food spawning were the next tasks tackled. The team was able to handle all collisions, growth, movement and rendering so the python was never transformed into a PyGame sprite.
3. After the base game was operational is when the menu was focused on. The menu was postponed until later since it was tied into the game settings, modes and UI improvements.
4. The final task was additional game modes and is where our team spent most of our time, additional game modes and settings required code refactoring and the gs object previously mentioned. The end result was the ability to add game features with ease.

## Conclusion

For this project, we implemented a fully functional classic Snake game. The newly unique version of the snake game has been created in Python. By making this project, we learned a new framework, PyGame and learned how to work with the proper software engineering methodologies. We faced many ups and downs while doing the project, but eventually we tested, fixed bugs, and updated our version of snake game.

## References


“Pygame Front Page.” *Pygame Front Page - Pygame v2.0.0.dev5 Documentation*, www.pygame.org/docs/.

Sommerville, I. *Software Engineering*. 10th ed., Pearson/Addison-Wesley, 2004.

“Python 3.8.2rc2 Documentation.” *3.8.2rc2 Documentation*, docs.python.org/3/.