Snake Game C++

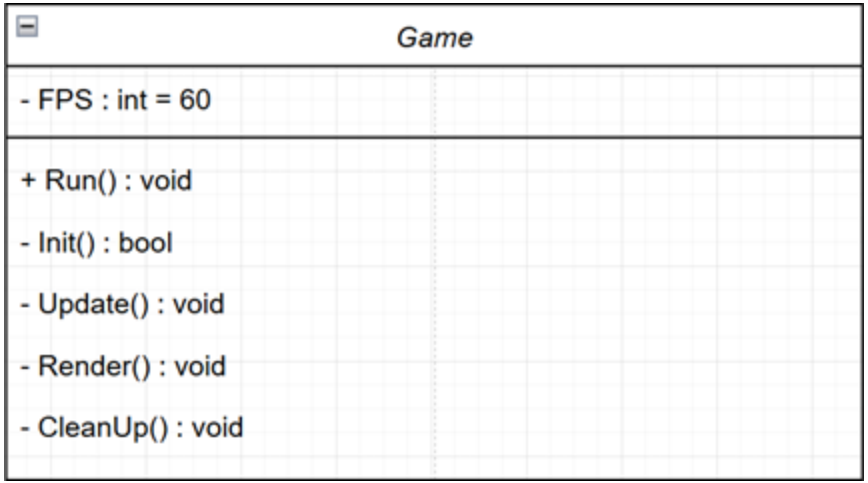
Create a classic snake game

Gameplay

- The snake starts as a small line or block, moving in a specific direction (up, down, left, or right).
- The player can control the snake's movement using arrow keys or another input method.
- When the snake eats food (represented as a small dot or pixel), it grows longer and moves faster.
- The goal is to eat as much food as possible without crashing into the walls or itself.

Part 1 - Foundation

- Import `SnakeGraphics` class
- Import `SnakeInput` class
- Create a game class according to the uml.
- Keep the frame rate
- Draw a main menu
- Move around in the main menu with the arrow keys



How to use the SnakeGraphics class

Start by defining how many tiles your world should consist of:

```
constexpr int WORLD_WIDTH = 60;
constexpr int WORLD_HEIGHT = 30;
```

During your games' initialization phase you initialize the SnakeGraphics object with the windows resolution and how many tiles the world shall consist of:

```
// Init graphics
Graphics = new SnakeGraphics(1024, 768, WORLD_WIDTH, WORLD_HEIGHT);

if (!Graphics->Init())
{
    std::cerr << "Failed to initialize graphics!" << std::endl;

    return false;
}
```

In the main loop you need to take care of the windows messages:

```
while (Graphics->UpdateWindowMessages())
```

How to use the SnakeGraphics class

Each frame you need to plot the tiles you want to have a specific color and character, otherwise they will be black.

```
Graphics->PlotTile(5, 5, 0, { 0, 255, 0 }, { 255, 255, 255 }, 'A');
```

Once all tiles have been plotted out, then you call the Render() function in Graphics.

```
Graphics->Render();
```

And of course, when closing the game, don’t forget to destroy the SnakeGraphics object:

```
delete Graphics;
```

How to use the SnakeInput class

The SnakeInput class is a Singleton. During your games' initialization phase you initialize SnakeInput by calling its Init() function, and pass the SnakeGraphics object as a parameter.

```
// Init input
if (!SnakeInput::Init(Graphics))
{
    std::cerr << "Failed to initialize input!" << std::endl;

    return false;
}
```

You create the function that you want to receive the key down event:

```
void SnakeGame::KeyDownCallback(int Key)
{
    std::cout << "Key down: " << Key << std::endl;
}
```

Then during your init phase you bind that function so it gets the key down events:

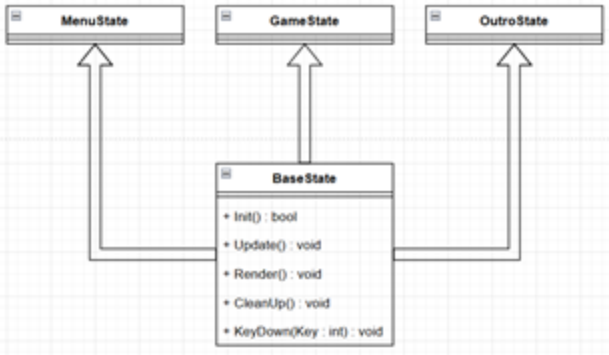
```
SnakeInput::AddKeyDownCallback(std::bind(&SnakeGame::KeyDownCallback, this, std::placeholders::_1));
```

Don't forget to cleanup SnakeInput:

```
SnakeInput::CleanUp();
```

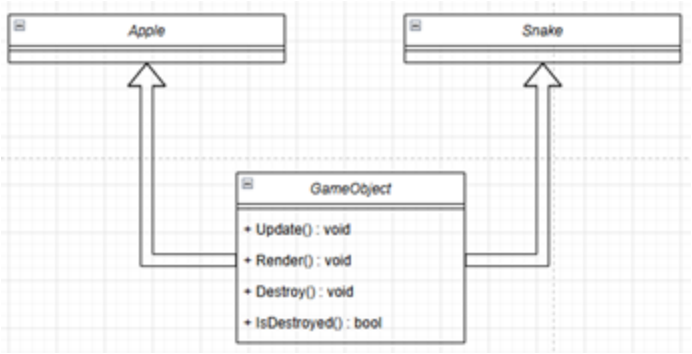
Part 2 - The State Machine

- Create a state machine according to the uml.
- All states must have their `Init` function called when activated.
- All states must have their `CleanUp` function called when deactivated.
- Only active states have their `Update`, `Render` and `KeyDown` functions called.
- When pressing “Start” in the menu, the `GameState` will be next active state.



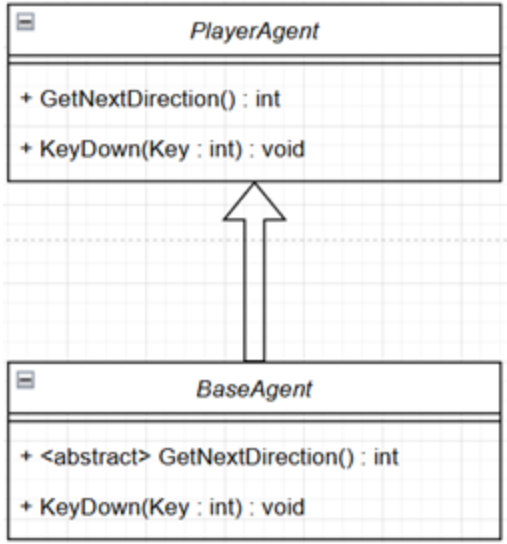
Part 3 - Game Objects

- Create a `World` class.
- Create a suitable size level and render it.
  - Load the level from a text file.
- Render the level.
- Create a `GameObject` class.
- Create a `Apple` class which inherits from `GameObject`
- Create a `Snake` class which inherits from `GameObject`
- The `World` class must manage the GameObjects by:
  - Creating them
  - Updating them
  - Rendering them
  - Destroying them
  - And whatever more you find useful



Part 4 - Game Play

- Render the snake
- Render the apple
- Have the snake move in a forward direction
- Create a **`BaseAgent`** class
- Create a **`PlayerAgent`** class which inherits from **`BaseAgent`**.
- The **`World`** class:
  - Creates the **`PlayerAgent`** class.
  - Provides the **`PlayerAgent`** with keyboard inputs.
  - Sends the **`PlayerAgent`** to the **`Snake`** Class (Like giving it a brain.)
- Pressing the keys makes the snake change direction



Part 5 - Completing the Game

- Create a collision system
  - Have the world class loop through the game objects and check if they collides with walls, or with each other.
  - Add a “`void OnCollision(GameObject* OtherObject)`” function into the `GameObject` class. This function will be called when a collision occurs.
- `OnCollision` between `Snake` and `Apple`:
  - The Apple gets destroyed
  - The Snake grows longer and moves faster.
  - A new Apple is created.
- `OnCollision` between `Snake` and `Snake` (Tail?) or between `Snake` and a `Wall`:
  - It is game over and the active state moves to the outro.
- Create a outro where you can see your score.
- Make it possible to leave the outro and go to the main menu.
- If the snake eats 10 apples it will proceed to the next level
- Make more levels.

Part 6 - Local Multiplayer

- Have the option to choose two players in the main menu
- Add another snake that can be controlled with different keys on the keyboard
- Add two multiplayer game modes
  - Co-op
    - Snakes are trying together to complete the levels.
    - If one snake dies, it is game over
  - Battle
    - Random level with a unlimited amount of apples.
    - If one snake crashes, then the other snake is the winner.

Part 7 - AI

- Add option to choose what players are human players and what players are AI players.
- Update the base agent class to receive information about the level.
- Create an `AIAgent` class.
- Implement one or several suitable AI algorithms to the `AIAgent` class to control the snake. Examples can be the following:
  - A-star
  - Minimax
  - Dynamic Danger Zone Mapping
  - Greedy Algorithm with Safety Checks
  - Tail-Chasing Strategy

```cpp
class BaseAgent
{
public:
    enum Direction
    {
        None = -1,
        Right = 0,
        Down = 1,
        Left = 2,
        Up = 3
    };

    struct LevelData
    {
        // Level is a 1D array of size Width * Height, where Level[i] = 0 is empty space, Level[i] = 1 is a wall
        bool* Level = nullptr;

        // Width and Height of the level
        int Width = -1;
        int Height = -1;

        // Position of players snake body, begin with the head.
        std::vector<std::pair<int, int>> PlayerSnake[2];

        // What player id the agent is
        int PlayerId = -1;

        // If the game is a cooperate game or not.
        bool Cooperate = false;
    };

    // Get the next direction the snake should go in (0 = right, 1 = down, 2 = left, 3 = up)
    virtual Direction GetNextDirection(const LevelData* InLevelData = nullptr) = 0;
};
```

Part 8 - Libraries

- Put the `AIAgent` class in a library which will be loaded at start.
- Borrow your friends AI libraries and have them compete against each other.