SP Project Milestone 1

Team: GEN8

Project: Pacman

Dr. Marwah Hilaly

20191700841 – عبد العزيز مصطفي عبد العزيز محمد

20191700794 – يوسف ناصر صابر بيلاطس

20191700269 – زياد محمد حسين خبيز

20191700282 – ستيفن سامح سليمان عطا الله

20191700493 – محمد أبو الحجاج محمد خليل

20191700160 – ايمن حسن توفيق عبد الحافظ

First and foremost, we created classes for some of the menus in GUI (Single player menu – Multiplayer menu – Menu).

Each of the created **menu classes** has the following functions:

1. **The constructor**:
   - In the constructor we have created a sf::text for each button (item) in the menu then we set their position to be centered.
2. `void MoveUp();`
   - It changes the current selected item to the one **above** it.
3. `void MoveDown();`
   - It changes the current selected item to the one **beneath** it.
4. `void draw(RenderWindow &window);`
   - @param it takes the window as a parameter by reference to draw on it.
   - It draws all the items (text buttons) on the screen after clearing the window with a dark blue color.

Dot class:

1. `sf::CircleShape getDot();`
   - @return a sf::CircleShape containing the **dot** entity.
2. `sf::CircleShape getSuperdot();`
   - @return a sf::CircleShape containing the **super dot** entity.

Animator class:

1. `void addFrame(sf::IntRect picture);`
   - @param sf::IntRect to define the position and the size of a new frame in the animation.
   - It pushes the new frame into the frames vector.
2. `bool isPlaying();`
   - @return a boolean indicating whether the animation is playing when the function is called or not.
3. `void setAnimation(sf::Time animationDuration, bool loop);`
   - @param sf::Time indicates the total time duration of displaying one cycle of the animation.
   - @param Boolean indicates whether the animation should be repeated or not.
   - It sets all the members of the class to their corresponding values.
4. `void update(sf::Time delta);`
   - @param the time the class should be updated with at every iteration of the game loop.
   - It updates the current frame to a new frame based on the time passed to the function.
5. `void animate(sf::Sprite &sprite);`
   - @param sf::Sprite passed by reference to set the sprite's IntRect with the current frame.

Pacman class:

1. `bool isDying();`
   - @return whether Pacman at that instance of time is **dying** or not.
2. `bool isDead();`
   - @return whether Pacman at that instance of time is **dead** or not.
3. `void makeDie();`
   - It checks if Pacman is neither dead nor dying then if such condition is met, it makes him die.
4. `void update(sf::Time delta);`
   - @param the time the class should be updated with at every iteration of the game loop.
   - If Pacman is dying, it displays the dying animation. If Pacman is neither dead nor dying, it displays the running animation. It also updates the animator instance with the sf::Time delta and handles when the dying animation ends.
5. `void draw(sf::RenderWindow &window);`
   - @param it takes the window as a parameter by reference to draw on it.
   - It draws Pacman on the window if he's not dead.
6. `void setPosition(float x, float y);`
   - @param set the position on X-axis
   - @param set the position on Y-axis
   - it sets the sf::Sprite `Pac-Man vision`'s position to the new coordinates.
7. `void setSpeed(float newSpeed);`
   - @param controls the speed of the movement.
8. `float getSpeed();`
   - @return Pacman's speed at that instance of time.

Maze Class:

The Class is inherited from `sf::Drawable,` as the maze is a drawable object.

Functions:

1. `void loadLevel(std::string file);`
   - @param **the level image file name** loads the level image and iterates on it pixel by pixel and add the pixels.
   - Colors to the mazeData vector as they are defined to their Entities.

2. `sf::Vector2i getPacmanPosition() const;`
   - @return Pacman's position.

3. `sf::Vector2i getBlinkyPosition() const;`
   - @return Blinky's position.

4. `sf::Vector2i getPinkyPosition() const;`
   - @return Pinky's position.

5. `sf::Vector2i getInkyPosition() const;`
   - @return Inky's position.

6. `sf::Vector2i getClydePosition() const;`
   - @return Clyde's position.

7. `inline unsigned int positionToIndex(sf::Vector2i position) const;`
   - @param a `sf::Vector2i` carries a position in the maze.
   - @return an unsigned int the index of the position in the mazeData vector.

8. `inline sf::Vector2i indexToPosition(unsigned int index) const;`
   - @param an unsigned int carries the index in the mazeData vector.
   - @return a `sf::Vector2i` the position of the index in the maze.

9. `void draw(sf::RenderTarget& target, sf::RenderStates states) const;`
   - @param `sf::RenderTarget` specifies the target is to be drawn
   - @param `sf::RenderStates` specifies the target states (i.e. the blend mode, the transform, the texture, and the shader).
   - Overriding the draw method as the maze is an inherited class from `sf::Drawable`
   - It iterates over the mazeData vector and draws the dots and the super dots, which the pacman have not eaten yet.

Bonus Class:

1. `Bonus(sf::Texture& texture);`
   - @param a `sf::Texture` in which the fruit is drawn.
   - Sets the origin and the TextureRect of the fruit.

2. `void drawBonus(sf::RenderWindow& window, float x, float y);`
   - @param `sf::RenderWindow` in which the bonus to be drawn.
   - @param `float x` the position of the bonus on the x-axis.
   - @param `float y` the position of the bonus on the y-axis.
   - It draws the Bonus on the window.

Ghost class:

1. **In the constructor** `Ghost(sf::Texture &texture, int type);`
   - @param sf:: Texture, sprite sheet that contains all the texture of game objects.
   - @param define the type of the ghost (Pinky, Inky, Blinky, Clyde)
   - Adding the corresponding frames to both of the two animators (animator for ghosts in their smart state when they attack Pacman and another animator for ghosts in their weak state in which they escape from Pacman.
2. `void setSpeed(float newSpeed);`
   - @param to update the ghost's speed.
3. `float getSpeed();`
   - @return the ghost's speed at that instance of time.
4. `bool isBlue();`
   - @return a boolean indicates whether the ghost's color is blue or not. In other words, whether the ghost is in the weak state or not.
5. `void makeBlue(sf::Time duration);`
   - @param the time duration to make the ghost blue (weak) for.
   - It makes the ghost weak for that period of time.
6. `void update(sf::Time delta);`
   - @param the time the class should be updated with at every iteration of the game loop.
   - If the ghost is blue, it displays the blue animation. Otherwise, it displays the default animation of the ghost. It also updates the animator instance with the sf::Time delta and handles when the blue animation ends.
7. `void draw(sf::RenderWindow &window);`
   - @param it takes the window as a parameter by reference to draw on it.
   - It draws the ghost on the window.
8. `void setPosition(float x, float y);`
   - @param to set the position on X-axis.
   - @param to set the position on Y-axis.
   - It sets the sf::Sprite `ghost's` position to the new coordinates.

Functions to be added:

1- Enhancing the maze visual:
   - The walls are going to be rounded, smoother, and unfilled.
2- Navigation logic:
   - Using graph algorithms:
     - Breadth First Search and A* that control the movement of ghosts.

Classes to be added:

1- Stages:
   - 3 different stages using one common class between them.
   - Each stage represents a difficulty.
2- Multiplayer mode through Lan network:
   - Two players through network with different rules from the classic game.
   - Timed based and weighted dots.
3- Scoreboard:
   - Scoreboard will save all previous scores for users.
4- Sounds settings:
   - Add sound to the background of the game.
5- In-Game statistics:
   - Providing live statistics for user during playing (score- time- lives).