“Pac-Man Desktop Port” Component Design

[Strange_Team_Name_Here]
(Rishi Basdeo, Ryan Tarr, Akshay Antony, Shreyas Misra)

Character Class

Developer**:** Rishi Basdeo

Description

The character class is used as the basis of the Pac-Man and Ghost classes. This class will define the motion of the character, as well as the basis for animation. The class will track current motion, call a function from the map to determine whether the current intended motion is achievable, and if so, will update the position of the object accordingly. (A special case will be needed for when moving through the “tunnel” connecting the left and right sides of the map.) The animation will be a stored array of images which will be iterated through at a predetermined interval, allowing the characters to “move” in place (ie. opening and closing Pac-Man’s mouth and moving the ghosts’ eyes). The motion of the character will be determined by the x and y positions of the character.

Member Variables

enum **direction** { up, down, left, right };
double **velocity**
double **position**[] = { x, y }
double **home_position**[] = { x, y }
yspng **alive_animation**[]
yspng **death_animation**[]
int **animation_index**
bool **is_alive**

Member Functions

void **loadImages**() - imports animation images
void **move**(*bool isFeasible(double* position*[], direction* target_direction*)*) - updates position based on *isFeasible* function from the map class
void **display**() - calls iterate_animation to show appropriate frame
void **iterate_animation**() - updates animation_index, checks is_alive to choose correct animation
bool **get_alive**() - returns whether the character is alive
void **kill_character**() - changes alive state
direction **get_direction**() - gets current direction
void **set_direction**(*direction* input_direction) - sets the desired direction of the character
void **get_position**(*double* position_array[]) - inserts character’s current position in position_array
void **setToHomePos**() - returns the character to its home position and sets default direction

Unit Tests

1. Direction Test- sets the direction to an arbitrary value and checks to make sure that the value is correct after unconstrained movement
2. Collision Test- repeats the direction test with constraints on direction using pseudo isFeasible function
3. Import Test - checks to make sure that animations are imported correctly

Pac-Man Class
Developer**:** Rishi Basdeo

Description
The Pac-Man class inherits from the character class and is controlled by the player. This class further integrates life tracking as well as checks for collisions with Pac-Pellets and ghosts. When defeated, it will respawn the character and trigger a reset that will allow for the ghosts and map to reset if necessary. It will take keyboard input for motion and pass the score to the map.

Member Variables
int **remaining_lives**
int **score** *(Moved to Scoreboard Class)*
bool **scatter_mode**

Member Functions
int **get_score**() - called by the map to get the updated score*(Moved to Scoreboard Class)*
bool **get_scatter_mode**() - called by the ghost class to determine which mode Pac-Man is in currently
void **set_scatter_mode**(*bool* mode) - sets the scatter mode to *mode*
int **increase_score**(int value) - increases the score by *value* amount *(Moved to Scoreboard Class)*
bool **check_collision**(double ghost_pos[]) - checks whether the position is the same as ghost_pos
void **update_lives**(*int* lives) - increases the number of lives by *lives*
void **get_lives**() - returns the number of lives, to be called by the map for illustration

Unit Tests
1. Interaction test- changes to scatter mode and tests to make sure that check_collision acts accordingly
2. Score Test- updates score and checks to make sure the value is correct *(Moved to Scoreboard Class)*
3. Life Test- updates lives and checks to make sure the value is correct

Ghost Class

Developer: Ryan Tarr

Description

The Ghost class inherits from the character class. Movement of a Ghost is determined by its target location, which is dependent on Pac-Man's current position. The movement behavior of a ghost is dependent on the member variable *movement_mode*. When this variable is set to CHASE, the Ghost moves toward the *chase_target* position

Member Variables

enum **movement_mode** {CHASE, SCATTER, FRIGHTENED}
double **chase_target**[] = { x, y }
const double **scatter_target**[] = { x, y }
const double **scatter_loop**
const int **scatter_loop_size**
int **dot_counter**
int **dot_threshold**
bool **leave_ghost_house**
static const double **NORMAL_SPEED**
static const double **FRIGHTENED_SPEED**

Member Functions

void **update_target**(double[] new_target) - updates target position of Ghost
bool **collide_pacman**() const - check to see if Ghost collided with Pac-Man
void **reset**() - reset some member variable values and move Ghost back to start position
int **get_turn_direction**() const - determine new direction for Ghost

Unit Tests

1. Movement Mode Test- ensure that each form of movement works appropriately (can switch between these using input keys for testing)
2. Leave Ghost House Test- The conditions for when the Ghosts can leave the ghost house vary between levels and when Ghosts are revived during levels (use input keys to kill/revive Ghosts in game and make sure they leave the Ghost house under proper conditions)
3. Chase Mode Targets- click on tiles in the map and verify that the Ghosts move toward their correct assigned target positions in chase mode

Animation

Developer: Akshay Antony

Description

This section of the code deals with all rendering in the game. For instance, functions for rendering the map, pacman, ghosts in positions. During game time the locations sent from the path planning and food collection will be rendered real time with appropriate animations.

Member functions

1. **void AnimationInitialize**(filename map, filename pacman, filename ghosts)
   Reads the png of map, pacman and ghosts. Resizes the png's to required pixel size. Renders the ghosts, pacman at the initial position before the game begins.
2. void **RenderGhosts**(vector<int> new_pos, int ghost_id)
   Takes the ghost id and new position from path planning and renders the ghosts to that position using some animation
3. void **RenderPacman**(vector<int> new_pos)
   Takes the new pacman position from path planning and renders the pacman with some animations which includes animations for collecting food.

Unit Tests

1. Render check- Will render the ghosts and pacman at the four corners of the screen, to verify the gl draw functions
2. Movement check- Try to give a position outside the map to animate to, which should fail in the unit test
3. Collision check- Make a function which tries to render two ghosts at the same pixel, which will fail the unit test.

Developer - Shreyas

```
class Consumable {
/*
    This is the class that controls the Consumables like the pellets.
    The pellets are distributed across the map and hence we need to store
the location and also store the state.
    0 -> not eaten 1-> eaten
*/
private:
    vector<int> loc;
    bool isEaten;
    bool power;
public:
    Consumable();
    bool getIsEaten(); //  returns whether the pellet is eaten or not
    vector<int> getLocation(); // return location of pellet
    void setIsEaten(bool); // set the pellet to eaten (or not eaten)
    void setLocation(int, int); // set location of the pellet
    bool getPower(); // check whether the pellet has a power or not
    void Draw(); // draw the pellets
};
class ScoreBoard {
/*
    This class controls the scoring mechanism for the game.
    It also stores the score to a text file which can be loaded in the
menu to show the high scores with the player name.

    Unit Tests for this class --
    1) Set Score Test - Setting and getting random scores
    2) Save Score - Saving the random scores to a file
    3) HighScore Test - Reading high scores from the file and displaying
it in the menu.
*/
private:
    long long int GameScore = 0;
    std::string highScoreFileName;
    std::string playerName; // from TextInput class
public:
    long long int GetScore(); // returns GameScore
    void SetScore(int); // Callable by PacMan to set score
    void SaveScore(); // Save the game score to the file
    ReadScores(); // Read the file containing the scores
    void setPlayerName(std::string);
    void drawScore();
    void displayHighScores(); };
class Map{
/*
    This class is responsible for generating the map and also for working
with the pellets and the locations of the pellets and ghosts.
    Reads the mapData from a file. The one shown below is a sample only.

    Unit tests:
    1) Map generation test - Generate random maps from file
    2) Pellet test - Check pellet and power pellet locations
    3) mapData test - Test the getMapBlock function to check if it is
returning the correct value.
*/
private:
    std::string fileName;
    std::vector<Consumable> pellets;
    std::vector<Consumable> powerPellets;
    std::vector<int> map;
    const char mapData[] = {
        "XXXXXXXXXXXXXXXX"
        "X  X           X"
        "X XXXX XXXXX XXX"
        "XXX X          X"
        "X   XX XXXXXXX X"
        "XXX  X XXXX XX X"
        "X X  X       X X"
        "X X  X XXXX XX X"
        "X XX X XXXX XX X"
        "X XX X X     X X"
        "X  X  X     XX X"
        "XXX  XX   X XX X"
        "X X XX XXXXXXX X"
        "X X  X XX X XX X"
        "XP             X"
        "XXXXXXXXXXXXXXXX"
    };
public:
    Map(); // To initialize the map
    void readMapFile(std::string);
    void RenderMap(const char[] mapData, int hei, int wid, const double
blockWid); // Takes the mapData, height, width and block width to render
the map
```

```
    unsigned int getMapBlock(const char[] mapData, int wid, int hei, int x, int y); // Gets the corresponding block at location (x,y) from the char array and returns as an int
    bool checkLocForPellet(int x, int y);
    bool checkLocForPowerPellet(int x, int y);
    bool checkLocForGhost(int x, int y);
    Consumable getPellet(int x, int y); // Helper function so that the PacMan class has access to the pellet at a particular location
    void Draw();
    ~Map(); // Clean Up everything
};
```