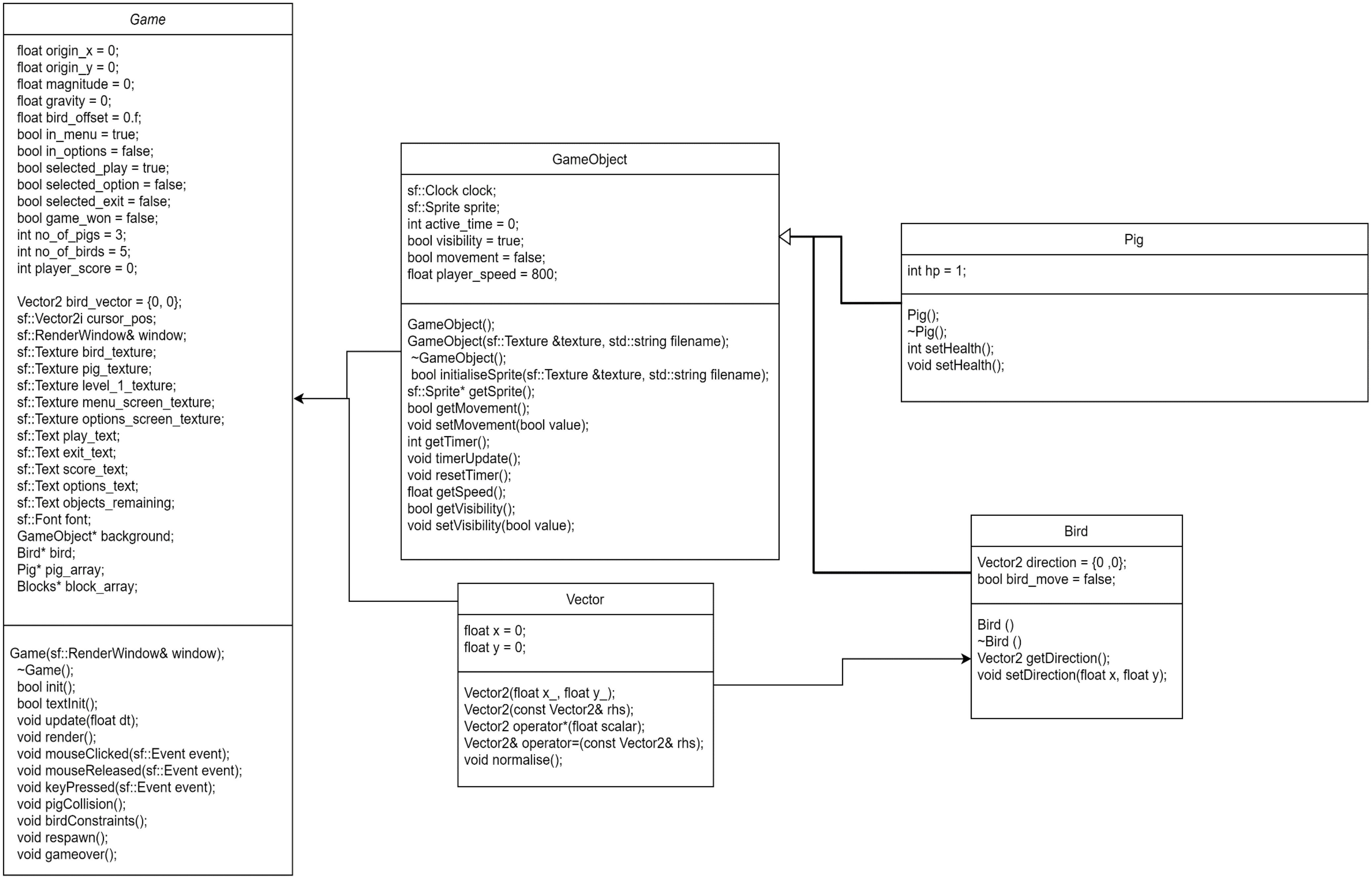
UML Diagrams (old and then new)



Game

```
float origin_x = 0;
float origin_y = 0;
float magnitude = 0;
float gravity = 0;
float bird_offset = 0.f;
bool in_menu = true;
bool in_options = false;
bool selected_play = true;
bool selected_option = false;
bool selected_exit = false;
bool game_won = false;
int no_of_pigs = 3;
int no_of_birds = 5;
int player_score = 0;

Vector2 bird_vector = {0, 0};
sf::Vector2i cursor_pos;
sf::RenderWindow& window;
sf::Texture bird_texture;
sf::Texture pig_texture;
sf::Texture level_1_texture;
sf::Texture menu_screen_texture;
sf::Texture options_screen_texture;
sf::Text play_text;
sf::Text exit_text;
sf::Text score_text;
sf::Text options_text;
sf::Text objects_remaining;
sf::Font font;
GameObject* background;
Bird* bird;
Pig* pig_array;
Blocks* block_array;
```

```
Game(sf::RenderWindow& window);
 ~Game();
 bool init();
 bool textInit();
 void update(float dt);
 void render();
 void mouseClicked(sf::Event event);
 void mouseReleased(sf::Event event);
 void keyPressed(sf::Event event);
 void pigCollision();
 void birdConstraints();
 void respawn();
 void gameover();
```

GameObject

```
sf::Clock clock;
sf::Sprite sprite;
int active_time = 0;
bool visibility = true;
bool movement = false;
float player_speed = 800;
```

```
GameObject();
GameObject(sf::Texture &texture, std::string filename);
 ~GameObject();
 bool initialiseSprite(sf::Texture &texture, std::string filename);
sf::Sprite* getSprite();
bool getMovement();
void setMovement(bool value);
int getTimer();
void timerUpdate();
void resetTimer();
float getSpeed();
bool getVisibility();
void setVisibility(bool value);
```

Pig

```
int hp = 1;
```

```
Pig();
~Pig();
int setHealth();
void setHealth();
```

Bird

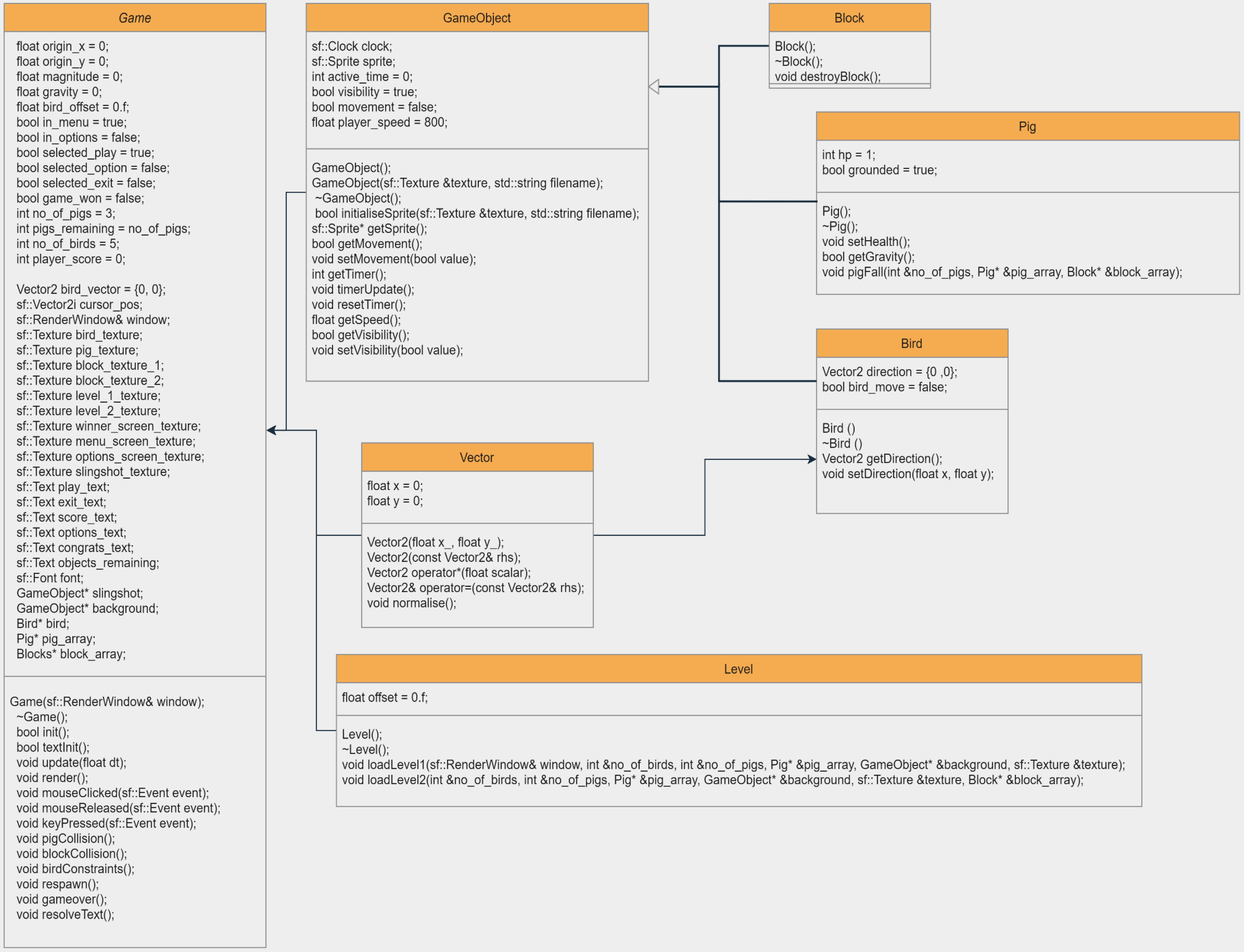
```
Vector2 direction = {0 ,0};
bool bird_move = false;
```

```
Bird ()
~Bird ()
Vector2 getDirection();
void setDirection(float x, float y);
```

Vector

```
float x = 0;
float y = 0;
```

```
Vector2(float x_, float y_);
Vector2(const Vector2& rhs);
Vector2 operator*(float scalar);
Vector2& operator=(const Vector2& rhs);
void normalise();
```



Game

```
float origin_x = 0;
float origin_y = 0;
float magnitude = 0;
float gravity = 0;
float bird_offset = 0.f;
bool in_menu = true;
bool in_options = false;
bool selected_play = true;
bool selected_option = false;
bool selected_exit = false;
bool game_won = false;
int no_of_pigs = 3;
int pigs_remaining = no_of_pigs;
int no_of_birds = 5;
int player_score = 0;

Vector2 bird_vector = {0, 0};
sf::Vector2i cursor_pos;
sf::RenderWindow& window;
sf::Texture bird_texture;
sf::Texture pig_texture;
sf::Texture block_texture_1;
sf::Texture block_texture_2;
sf::Texture level_1_texture;
sf::Texture level_2_texture;
sf::Texture winner_screen_texture;
sf::Texture menu_screen_texture;
sf::Texture options_screen_texture;
sf::Texture slingshot_texture;
sf::Text play_text;
sf::Text exit_text;
sf::Text score_text;
sf::Text options_text;
sf::Text congrats_text;
sf::Text objects_remaining;
sf::Font font;
GameObject* slingshot;
GameObject* background;
Bird* bird;
Pig* pig_array;
Blocks* block_array;
```

```
Game(sf::RenderWindow& window);
 ~Game();
 bool init();
 bool textInit();
 void update(float dt);
 void render();
 void mouseClicked(sf::Event event);
 void mouseReleased(sf::Event event);
 void keyPressed(sf::Event event);
 void pigCollision();
 void blockCollision();
 void birdConstraints();
 void respawn();
 void gameover();
 void resolveText();
```

GameObject

```
sf::Clock clock;
sf::Sprite sprite;
int active_time = 0;
bool visibility = true;
bool movement = false;
float player_speed = 800;
```

```
GameObject();
GameObject(sf::Texture &texture, std::string filename);
 ~GameObject();
 bool initialiseSprite(sf::Texture &texture, std::string filename);
sf::Sprite* getSprite();
bool getMovement();
void setMovement(bool value);
int getTimer();
void timerUpdate();
void resetTimer();
float getSpeed();
bool getVisibility();
void setVisibility(bool value);
```

Block

```
Block();
~Block();
void destroyBlock();
```

Pig

```
int hp = 1;
bool grounded = true;
```

```
Pig();
~Pig();
void setHealth();
bool getGravity();
void pigFall(int &no_of_pigs, Pig* &pig_array, Block* &block_array);
```

Bird

```
Vector2 direction = {0 ,0};
bool bird_move = false;
```

```
Bird ()
~Bird ()
Vector2 getDirection();
void setDirection(float x, float y);
```

Vector

```
float x = 0;
float y = 0;
```

```
Vector2(float x_, float y_);
Vector2(const Vector2& rhs);
Vector2 operator*(float scalar);
Vector2& operator=(const Vector2& rhs);
void normalise();
```

Level

```
float offset = 0.f;
```

```
Level();
~Level();
void loadLevel1(sf::RenderWindow& window, int &no_of_birds, int &no_of_pigs, Pig* &pig_array, GameObject* &background, sf::Texture &texture);
void loadLevel2(int &no_of_birds, int &no_of_pigs, Pig* &pig_array, GameObject* &background, sf::Texture &texture, Block* &block_array);
```

Initially I thought I would make the game of a much lower scope, but I kept building upon the base layer. I needed to add more text handling as I wanted the text objects to be unique rather than constantly changing the strings and their positions. It helped keep the code tidier and easier to understand. That is also why I implemented the function for resolving text within the menu, it is not doing much, but is separated to not cluster up the rest of the code.

For the GameObject class I did not do many changes as I mostly was able to reuse the code I had written from previous assignments and could imagine how to structure most of it from the beginning.

Writing the Pig class, I changed directions to make some more slightly believable physics. Before implementing the Block class, I wanted to make some pigs as boss enemies with more health, but afterwards I decided to create function which gave them a gravitational pull instead. Once the blocks they stand on are gone, the pigs fall to the ground. The health I still left within the class in case I went back to the project later or would like to reuse it at some point.

At first, I was thinking that if I did create a second level, I would only reposition the existing object after resetting their visibility, but I wanted to handle that in a separate class. I am sure it could be managed more efficiently, but as an experimental class I believe it is still better to have as it is much easier to use and take out all the positioning functions and handling outside of the main Game class which would make the code much harder to read. I know that next time I should try and make sure to keep the text-based handling and management in a separate class as well.

I am still learning about when I should create a new class and when I should leave it as a simple object. For the slingshot object I left without a separate class as it has no interaction available, but I decided to move the Block objects outside since they possessed some functionality even if it was not major, but in my opinion still important for keeping the code neater.

Pseudocode

```
1. Check for the click of the mouse;
2. (If the bird is not moving and the mouse clicks on the bird):
    Move the bird to where the mouse is;
3. Check for the release of the mouse;
4. Calculate the vector path for the bird          vector = bird_origin – bird_position;
5. Calculate the power with magnitude              magnitude = sqrt(vector.x^2 + vector.y^2);
6. Normalize vector;
7. (If the bird is moving to the right):
        Move the bird sprite: bird->getSprite()->move(vector.x * game_speed * dt,
                                                      vector.y * game_speed * dt + gravity - power);
8. (If the bird has not hit the ground):
        Increase the value of the gravity          gravity += constant_value;
   Else:
        Respawn the bird;
        Reset gravity to 0;
```