ISTANBUL TECHNICAL UNIVERSITY

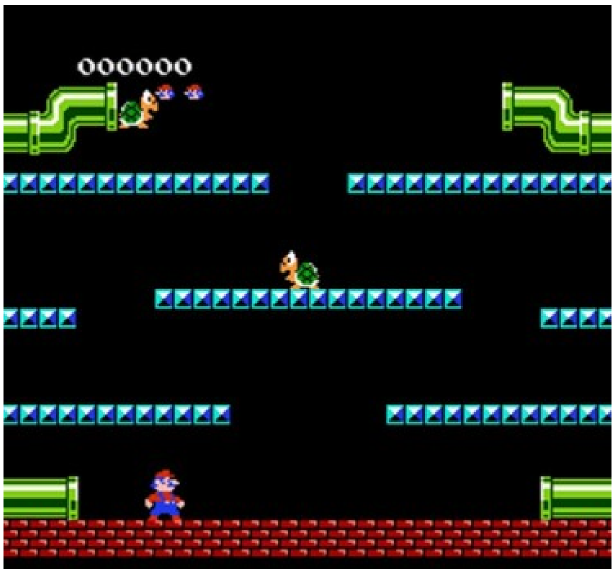
FACULTY OF ELECTRIC–ELECTRONICS

DEPARTMENT ELECTRONICS & COMMUNICATION ENGINEERING

EHB354E Object Oriented Programming

Term Project Report

The Classical Mario Platform Game



Fatih Sari

040190221

INTRODUCTION

In this project, we designed a version of the classic Mario game, an old retro arcade game, and coded it in C++ using object-oriented programming techniques. In the project, we worked on many important sub-topics such as inheritance, data protection, working with objects with private data, Polymorphism in object-oriented programming, which is the most important feature that we learned during the term and that distinguishes C++ from normal C, and we developed code. In addition to all these, we also used the link list data structer to make the objects we use more efficient. Thanks to this project, we have produced an efficient product by using C++, object-oriented programming and the SFML library, which is an important multimedia library, on a large scale. The main objectives of the project have been achieved, and there are some parts that have been developed from the bonus parts.

TEAM INFO

Team Name : partners_in_crime

```
Fatih Sari - 040190221          Roles: Planning/Analysis, Design, Implementation, Testing
Mehmet Soner Korucu - 040190225 Roles: Planning/Analysis, Design, Implementation, Testing
```

We worked together on most of the issues and edited the design. We divided the main code blocks into parts and made work divisions in order to progress faster. In the development section, which parts are made and how they are made will be explained in detail.

IMPLEMENTATION

1. Game Class
This class is the backbone of the game. It has methods for printing all the background elements of the game, defining the link list functions of the created objects, and controlling the Mario and Turtle collisions.

Attributes (private):

```
ObjectNode*_head;                 //Head specified to link objects to link list.
sf::Texture _floorTexture;        //The textures and sprites of floor, brick and
sf::Texture _brickTexture;        pipes are defined in this module.
sf::Texture _pipesBackground[2];
sf::Sprite _floor;
sf::Sprite _brick[7];
sf::Sprite _pipes[4];
```

Methods (public):

```
Game(sf::RenderWindow &window);                        //Constructor
void drawBackground(sf::RenderWindow& window);         //Set and draw background
void setBackground(sf::RenderWindow& window);          objects
bool onFloor(Object *obj);                             //Controls whether it comes
bool checkCollusion(Turtle* t, Mario* m, int& side);   into contact with solid objects
void AddObject(Object *obj);                           in the game.
void DeleteObject(Object *obj);                        //Checks if Mario has hit a
Object *getObject(int i);                              turtle and from which side
int  ObjectCount();                                    //Add, delete and get functions
                                                       for link list
```

2. Object Class
This class will keep attributes and methods for the base object class. Mario and Turtle classes explained below will derive from this class via inheritance. Below are minimum required attributes and methods.

Attributes (protected):

```
sf::Texture _textures[8];      //Textures, sprite and position vector is common for turtle
sf::Sprite _sprite;            and mario, So we initilized them in object class.
sf::Vector2f _pos;             //_heading tells which direction the object is facing
sf::RenderWindow* _window;     //isDead is a flag for live or dead object
int   _heading;                //walking used for fast texture changes for walking animation
bool isDead;
int   walking;
```

Methods (public):

```
Object(sf::RenderWindow* window);        //setPosition is set to object position; getPosition is
void setPosition(sf::Vector2f pos);      return the position of object.
sf::Vector2f getPosition();              //boundingBox returns covers of object like in
sf::IntRect boundingBox();               rectangular shape.
bool getIsDead();                        //move for lateral movement, fall is control dead
virtual void move(void) = 0;             scenario, jump for horizontal movement.
virtual void fall(void) = 0;
virtual void jump(bool down) = 0;
```

3. Mario Class

Attributes (private):

```
float     _vx;                           //_vx and _vy is a speed attributes for Mario and Turtle
float     _vy;
```

Methods (public):

```
Mario(sf::RenderWindow* window);         //CornerChecks is limiting Mario’s location for don’t
void CornerChecks();                     touching pipes. Because Mario cannot get into pipe.
void move();
void jump(bool down);                    //move, jump and fall methods is already coming from
void fall(void);                         object but inside the function, there is some difference
                                         between classes.
```

4. Turtle Class

Attributes (private):

```
float     _vx;
float     _vy;
```

Methods (public):

```
Turtle(sf::RenderWindow* window);
void move();
void jump(bool down);
void fall(void);
```

5. ScoreBoard Class

Attributes (protected):

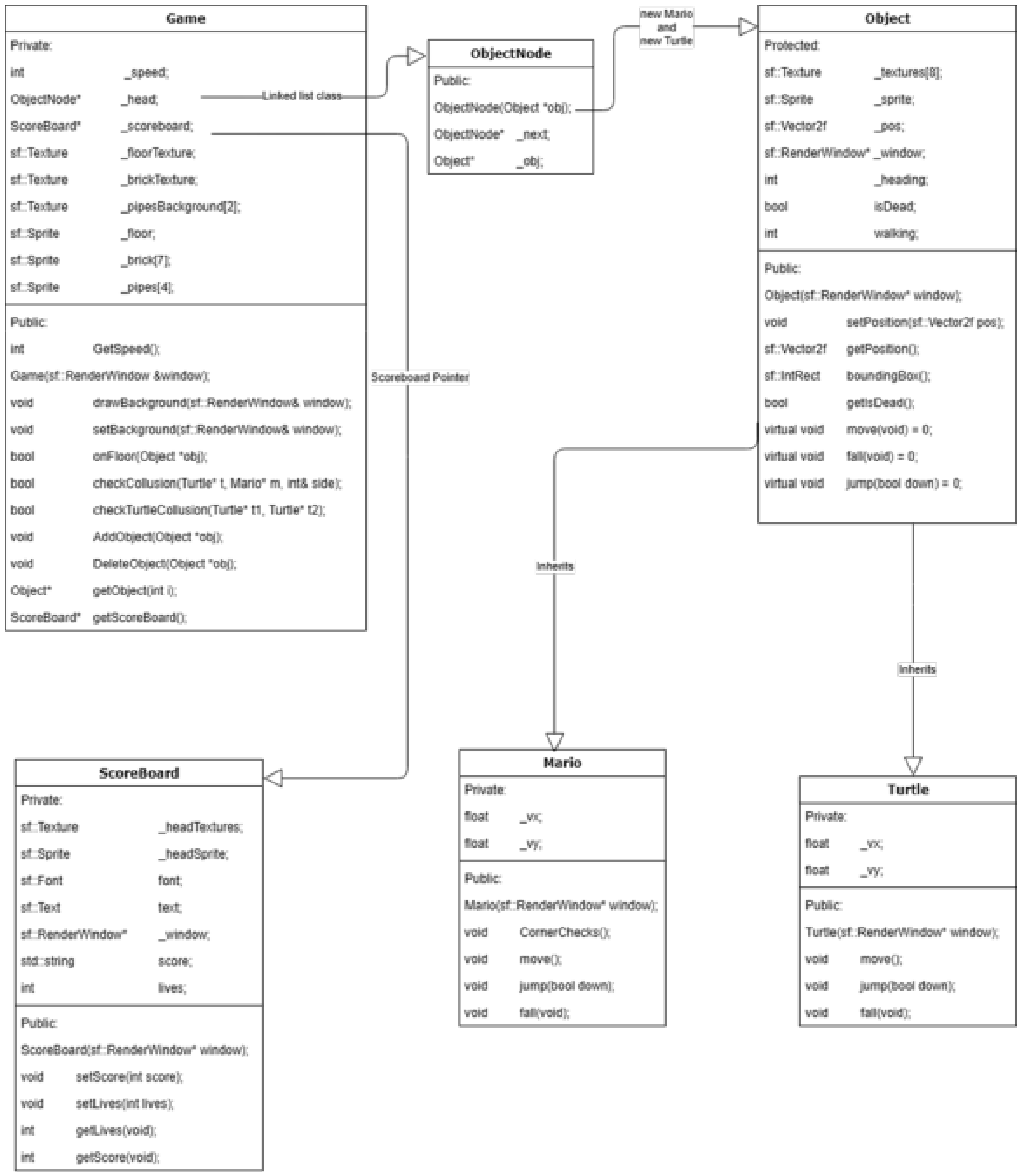
```
sf::Texture _headTextures;               //_headTextures and _headSprite is located for counted
sf::Sprite _headSprite;                  how much health Mario.
sf::Font font;
sf::Text text;
sf::RenderWindow* _window;
std::string    score;                    //current score
int  lives;                              //remaining life count for Mario.
```

Methods (public):

```
ScoreBoard(sf::RenderWindow* window);
void setScore(int score);
void setLives(int lives);                // sets and gets lives.
int       getLives(void);
int       getScore(void);
```

```
6. ObjectNode Class                          //this class is created for create
   Attributes (public):                      link list properly.
      Object *_obj;
      ObjectNode *_next;
   Methods (public):
      ObjectNode(Object *obj);
```

CLASS STRUCTERS FLOW CHART



```
Game
Private:
int             _speed;
ObjectNode*     _head;
ScoreBoard*     _scoreboard;
sf::Texture     _floorTexture;
sf::Texture     _brickTexture;
sf::Texture     _pipesBackground[2];
sf::Sprite      _floor;
sf::Sprite      _brick[7];
sf::Sprite      _pipes[4];
Public:
int             GetSpeed();
Game(sf::RenderWindow &window);
void            drawBackground(sf::RenderWindow& window);
void            setBackground(sf::RenderWindow& window);
bool            onFloor(Object *obj);
bool            checkCollusion(Turtle* t, Mario* m, int& side);
bool            checkTurtleCollusion(Turtle* t1, Turtle* t2);
void            AddObject(Object *obj);
void            DeleteObject(Object *obj);
Object*         getObject(int i);
ScoreBoard*     getScoreBoard();
```

```
ObjectNode
Public:
ObjectNode(Object *obj);
ObjectNode*   _next;
Object*       _obj;
```

```
Object
Protected:
sf::Texture          _textures[8];
sf::Sprite           _sprite;
sf::Vector2f         _pos;
sf::RenderWindow*    _window;
int                  _heading;
bool                 isDead;
int                  walking;
Public:
Object(sf::RenderWindow* window);
void            setPosition(sf::Vector2f pos);
sf::Vector2f    getPosition();
sf::IntRect     boundingBox();
bool            getIsDead();
virtual void    move(void) = 0;
virtual void    fall(void) = 0;
virtual void    jump(bool down) = 0;
```

```
ScoreBoard
Private:
sf::Texture          _headTextures;
sf::Sprite           _headSprite;
sf::Font             font;
sf::Text             text;
sf::RenderWindow*    _window;
std::string          score;
int                  lives;
Public:
ScoreBoard(sf::RenderWindow* window);
void     setScore(int score);
void     setLives(int lives);
int      getLives(void);
int      getScore(void);
```

```
Mario
Private:
float    _vx;
float    _vy;
Public:
Mario(sf::RenderWindow* window);
void     CornerChecks();
void     move();
void     jump(bool down);
void     fall(void);
```

```
Turtle
Private:
float    _vx;
float    _vy;
Public:
Turtle(sf::RenderWindow* window);
void     move();
void     jump(bool down);
void     fall(void);
```

DISCUSSION

In general, we did not encounter a very serious problem that cannot be solved. It took us some time to sort out Mario's inconsistent behavior in some collision situations. We've done a bit of work with Pipelar to eliminate teleporting to nonsensical locations in collision druums. In addition, using a link list delayed the completion of the project by about 2 days. We take great pleasure in presenting you a nice retro Mario platform game with minor bugs with the elimination of necessary controls and memory leaks. After evaluating our work, we would like to receive feedback from you in order to improve our future projects. We would like to state that we did not receive **ANY HELP FROM ANYONE** in this project, which we developed completely by ourselves.