CIS 29 FINAL PROJECT

PINK BEAN'S ADVENTURE

GROUP 5: BRIAN ASPINWALL | DARYL FOO | KEITH HUNG | WEIJING YONG

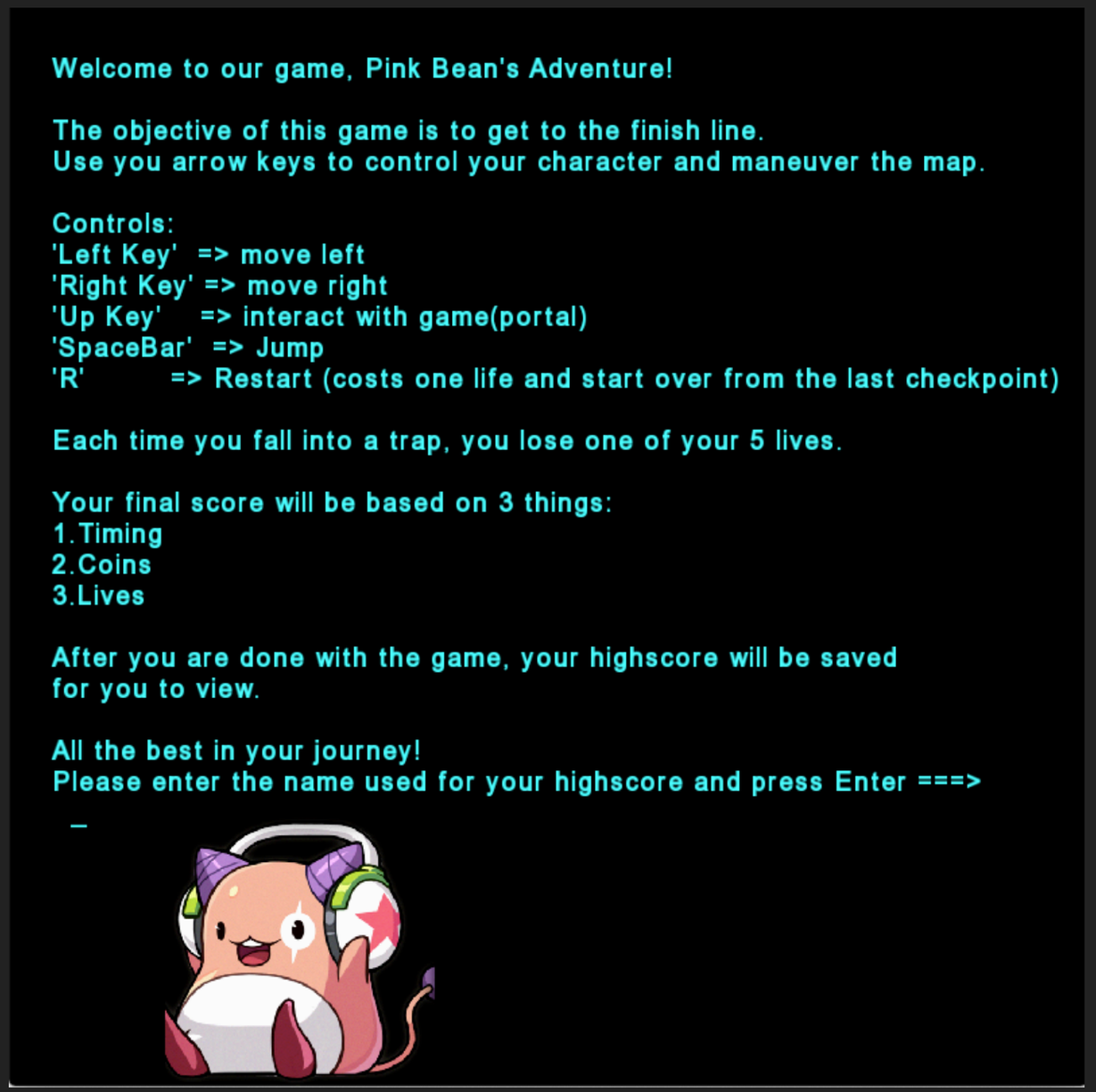
TABLE OF CONTENTS

PINK BEAN'S ADVENTURE

- Pink Bean is lost in a maze full of traps, coins, and power-ups
- The objective of the game is to help Pink Bean get to the finish line with the highest score possible
- The final score is calculated with:
  - 1. Time
  - 2. Coins
  - 3. Lives

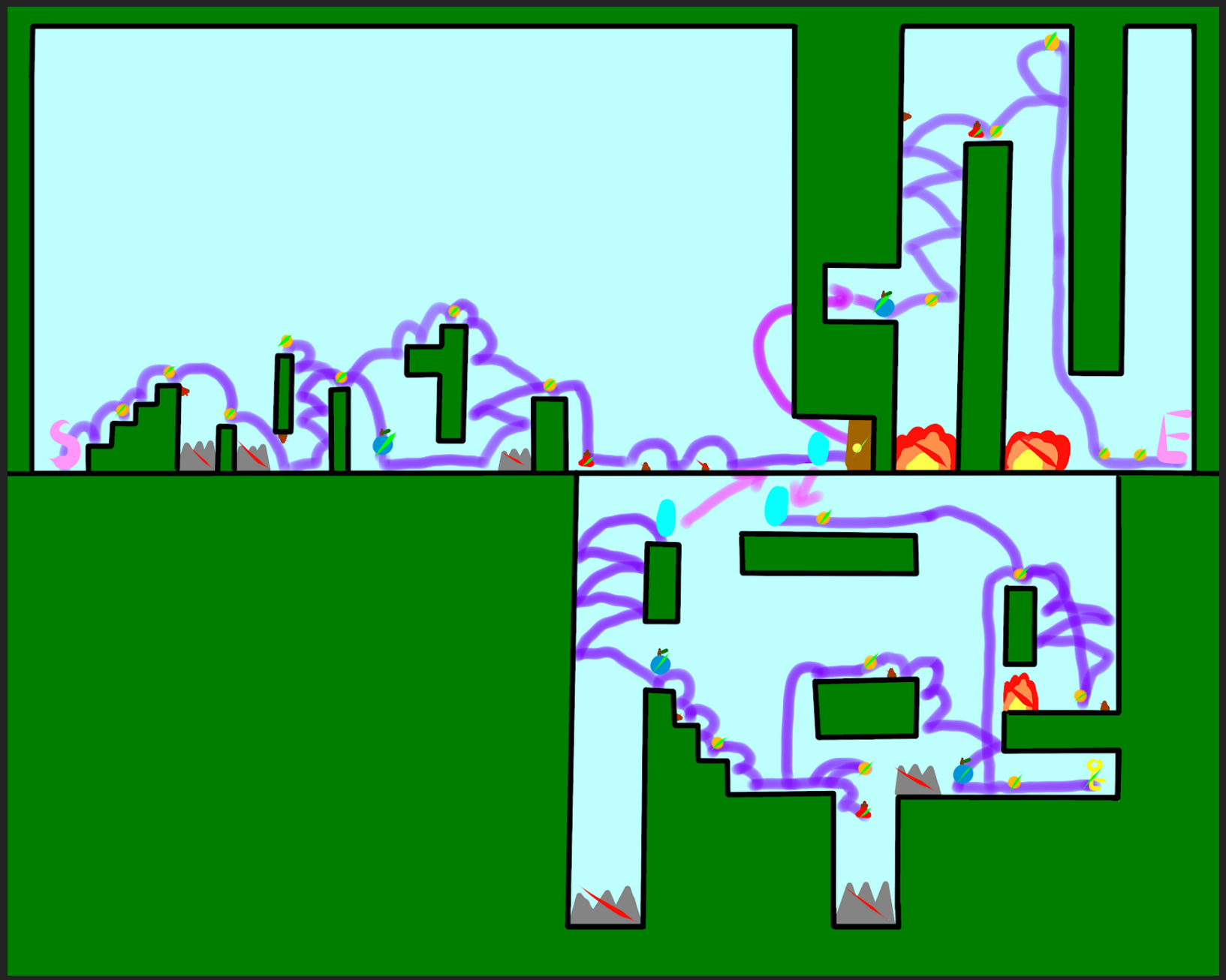
INTRODUCTION SCREEN

- The introduction screen provides instructions on gameplay and prompts player to enter their name for highscores



GAME MAP

- The ultimate guide to finishing the game



GAME DESIGN

Player

- body: sf::Sprite
- animation:: class Animation
- row : unsigned
- faceLeft : bool
- score : unsigned
- lives : unsigned
- speed : float
- velocity : v2f
- wallJumpWindowL : unsigned
- wallJumpWIndowR : unsigned
- canJump : bool
- jumpHeight : float
- hasKey : bool
- hasWon : bool
- hasLost : bool
- hasFinished : false
- atHighScore : false
- checkpoint : v2f

- Player(sf::Texture* texture, v2u imageCount, float switchTime, float speed, float jumpHeight)
- ~Player()
- GetPosition() : v2f
- GetOrigin() : v2f
- GetGlobalBounds() : sf::FloatRect
- getScore() : int
- getLives() : int
- getKeyStatus() : bool
- getWinStatus() : bool
- getLoseStatus() : bool
- getFinishStatus() : bool
- getCheckpoint() : v2f
- getAtHighscore() : bool
- GetCollider() : Class Collider
- setScore(int s) : void
- setWin(bool) : void
- setLose(bool) : void
- setFinished(bool) : void
- setCheckpoint(v2f cp) : void
- obtainedKey() : void
- Update(float deltaTime) : void
- Draw(sf::RenderWindow &window) : void
- OnCollision(v2f direction) : void
- addLives() : void
- loseLives() : void
- TP(v2f dest) : void
- TPPart1() : void
- TPPart2() : void
- TPPart3() : void
- TPEnd() : void
- TPHighscore() : void
- TPCheckpoint() : void
- Boost() : void
- Revert() : void

Collider

- body : sf::Sprite

- Collider(sf::Sprite &body)
- ~Collider()
- GetPosition() : v2f
- GetHalfSize() : v2f
- Move(float dx, float dy) : void
- CheckCollision(Collider other, v2f &direction, float push) : bool

Platform

- body : sf::Sprite

- Platform(sf::Texture* texture, v2f scale, v2f position)
- ~Platform()
- GetCollider() : class Collider
- Draw(sf::RenderWindow &window) : void

Bound::Error

- message : std::string

- Error(std::string text)
- ~Error()
- operator <<(std::ostream& out, const Error& error) : friend std::ostream

Portal

- body : sf::Sprite
- destination : v2f
- checkpoint : v2f

- Portal(sf::Texture* texture, v2f scale, v2f position, v2f dest)
- ~Portal()
- UpdatePlayer() : void
- GetGlobalBounds() : sf::FloatRect
- Draw(sf::RenderWindow &window) : void

Textures::Error

- message : std::string

- Error(std::string text)
- ~Error()
- operator <<(std::ostream& out, const Error& error) : friend std::ostream

Animation

- uvRect : sf::IntRect
- imageCount : v2u
- currentImage : v2u
- totalTime : float
- switchTime : float

- Animation(sf::Texture* texture, v2u imageCount, float switchTime)
- ~Animation()
- Update(int row, float deltaTime, bool faceLeft) : void

Font::Error

- message : std::string

- Error(std::string text)
- ~Error()
- operator <<(std::ostream& out, const Error& error) : friend std::ostream

Trap

- body : sf::Sprite

- Trap(sf::Texture* texture, v2f scale, v2f position, float rotation)
- ~Trap()
- GetGlobalBounds() : sf::FloatRect
- UpdatePlayer() : virtual void
- Draw(sf::RenderWindow &window) : void

Potion

- Potion(sf::Texture* texture, v2f position)
- ~Potion()
- UpdatePlayer() : virtual void override

Sound::Error

- message : std::string

- Error(std::string text)
- ~Error()
- operator <<(std::ostream& out, const Error& error) : friend std::ostream

Item

- body : sf::Sprite

- Item()
- Item(sf::Texture* texture, v2f position)
- ~Item()
- GetGlobalBounds() : sf::FloatRect
- UpdatePlayer() : virtual void
- Draw(sf::RenderWindow &window) : void

Key

- Key(sf::Texture* texture, v2f position)
- ~Key()
- UpdatePlayer() : virtual void override

Score

- name : std::string
- date : std::string
- score : int

- Score(std::string name, std::string date, int score)
- Score()
- operator>(const Score& other) const : bool
- operator<<(std::ostream& os, const Score& score) std::ostream&

Coin

- Coin(sf::Texture* texture, v2f position)
- ~Coin()
- UpdatePlayer() : virtual void override

Apple

- pos : v2f

- Apple(sf::Texture* texture, v2f position)
- ~Apple()
- UpdatePlayer() : virtual void override
- Refresh() : void

BACKDOORS

- Backdoors were used to facilitate testing of game functionalities during the development phase

```
//Back door commands
if(sf::Keyboard::Key::Q == event.key.code)
{
    player.Boost(); // 2x speed and jump
}
if(sf::Keyboard::Key::W == event.key.code)
{
    player.Revert(); // normal speed and jump
}
if(sf::Keyboard::Key::Num1 == event.key.code)
{
    player.TPPart1();
}
if(sf::Keyboard::Key::Num2 == event.key.code)
{
    player.TPPart2();
}
...
```

RANDOMNESS

- Clouds that float around in the background are generated randomly

```
rand(time(0));
```

```
unsigned numClouds{static_cast<unsigned>(rand() % 50 + 150)};
for(auto i = 0u; i < numClouds; i++)
{
    sf::Sprite tempCloud;
    float randomScale{static_cast<float>(rand() % 4 + 1)};
    float randomX{static_cast<float>(rand() % 12000)};
    float randomY{static_cast<float>((rand() % 9000) - 3000)};
    tempCloud.setOrigin(v2f(50.0f, 50.0f));
    tempCloud.setTexture(cloudTexture);
    tempCloud.setScale(v2f(randomScale, randomScale));
    tempCloud.setPosition(v2f(randomX, randomY));
    clouds.push_back(tempCloud);
}
```

USER INTERACTIVITY

- Both keyboard and mouse input are fed into the game
- Keyboard mainly used for navigating around, the mouse for interacting with objects

3RD PARTY CODE

- Animation and collision detection was referenced from Hilze Vonck's youtube channel
- www.youtube.com/channel/UC8C7ncaMYnXyu-pRU0S9FLg

GAME TIME

- Quick demonstration of Pink Bean’s Adventure

POLYMORPHISM

- All items (Coin, Apple, Key, Potion) are derived from the abstract base class Item

- A pure virtual function is implemented in Item to update the player with the type of item accordingly

```
virtual void UpdatePlayer(Player &player) = 0;
```

- The pure virtual function is overridden in derived classes of Item.

```
void UpdatePlayer(Player &player) override;

void Coin::UpdatePlayer(Player &player)
{
    player.setScore(player.getScore() + 100);
    body.setPosition(v2f(-1000.0f, 0.0f));
}
```

→ **Definition of UpdatePlayer() in class Coin**

NAMESPACE

- The class Error was used for handling exceptions
- Since there were different types of exceptions (e.g. audio, font, textures etc.) , namespaces are used to avoid name clashes of the class Error

```
namespace Bound
{
    class Error
    {
        private:
            std::string message;
        public:
            Error(std::string text) : message(text) { }
            friend std::ostream& operator<<(std::ostream& out, const Error& error);
    };
}
```

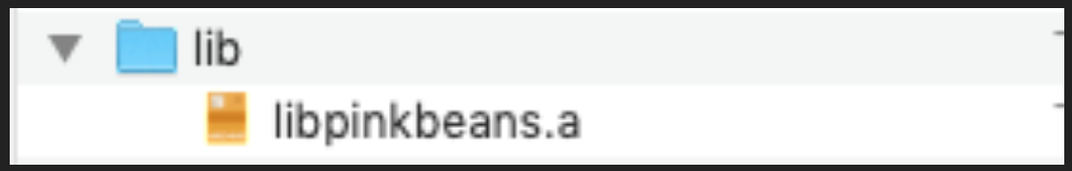
→ **class Error in namespace Bound**

```
namespace Textures
{
    class Error
    {
        private:
            std::string message;
        public:
            Error(std::string text) : message(text) { }
            friend std::ostream& operator<<(std::ostream& out, const Error& error);
    };
}
```

→ **class Error in namespace Textures**

LIBRARY

- All source code was converted to object files and archived into libpinkbeans.a



EXCEPTION HANDLING

- As mentioned, the class Error was used for exception handling
- 4 types of exceptions: out of bounds, texture load, font load, sound load

```
try
    {
        if((player.GetPosition().x < -3000.0f || player.GetPosition().x > 12000.0f) && (player.GetPosition().x != 13500.0f && player.GetPosition().x != 15000.0f))
        {
            throw Bound::Error("Out of bounds!");
        }
        if(player.GetPosition().y < -2500.0f || player.GetPosition().y > 6000.0f)
        {
            throw Bound::Error("Out of bounds!");
        }
    } catch(const Bound::Error& error)
    {
        std::cerr << error << std::endl;
        player.TPPart1();
    }
```

Out of bounds if player lies in specific range of coordinates

OVERLOADED INSERTION OPERATORS

- Insertion operators were overloaded to generate custom output to console or files

```
friend std::ostream& operator<<(std::ostream& os, const Score& score)
{
    os << std::endl  << "          " << score.name << std::string(13 -
        (score.name).length(), ' ') << score.date << std::string(10 -
        (score.date).length(), ' ') << score.score;
    return os;
}
```

Overloaded insertion operator for writing scores to file and screen

STL CONTAINERS (1) – VECTOR

- The STL vector container was used to store various objects throughout the game (e.g. Platforms, Coins, Apples etc.)

```
std::vector<Coin> coins;
// Part 1
coins.push_back(Coin(&coinTexture, v2f(975.0f, -200.0f)));
coins.push_back(Coin(&coinTexture, v2f(1485.0f, -400.0f)));
coins.push_back(Coin(&coinTexture, v2f(2175.0f, -200.0f)));
```

Pushing various objects into vectors

```
std::vector<Platform> platforms;
    // Part 1
    platforms.push_back(Platform(&platformTexture, v2f(30.0f, 50.0f),
       v2f(-1600.0f, 0.0f))); // Startwall
    platforms.push_back(Platform(&platformTexture, v2f(500.0f, 20.0f), v2f(0.0f,
       1100.0f))); // Floor
    platforms.push_back(Platform(&platformTexture, v2f(500.0f, 10.0f), v2f(0.0f, -3000.0f)));
        // Ceiling
```

STL CONTAINERS (2) – MULTISET

- The STL multiset Container was used to store Score objects for reading and writing highscores

```
std::multiset<Score, std::greater <Score> > scores;
```

Declaration of multiset scores

```
while(ifs >> score)
    {
        ifs.ignore(1, ';');
        getline(ifs, name, ';');
        ifs >> date;
        scores.insert(Score(name, date, score));
    }
```

Reading in from scores file and storing into multiset

Displaying scores from multiset container

```
std::multiset<Score, std::greater <Score> >::iterator itr = scores.begin();
    int numPrinted{0};
    while (itr != scores.end() && numPrinted < 5)
    {
        ofs << itr->score << ';' << itr->name << ';' << itr->date << std::endl;
        numPrinted++;
        std::advance(itr, 1);
    }
```

C++ 11 FEATURES (1, 2) – AUTO KEYWORD, RANGE BASED FOR LOOP

- The auto keyword was used in conjunction with range based for loops to “draw” objects to the screen

```
for(auto &platform : platforms)
{
    platform.Draw(window);
}
for(auto &coin : coins)
{
    coin.Draw(window);
}
```

→ **draw platforms and coins from its vector containers**

C++ 11 FEATURES (3)- BRACE INITIALIZATIONS

- List initializations were used for all variable initializations except for first line in for loops

```
static const float VIEW_HEIGHT{512.0f};
const std::string ResourcePath{""};
const std::string Sounds{"Sounds/"};
const std::string Sprites{"Sprites/"};
const std::string Fonts{"Font/"};
const std::string Texts{"Text/"};
```

→ **initializing file paths and view height**

```
bool upPressed{false};
bool statsRetrieved{false};
bool doorLocked{true};
```

→ **initializing variables used throughout the program**

```
int timeBonus{500 - timeTaken};
```

C++ 11 FEATURES (4) – INITIALIZER LIST IN CONSTRUCTORS

- Initializer list used for constructors to initialize members of classes

```
Portal::Portal(sf::Texture* texture, v2f scale, v2f position, v2f dest)
    : destination(dest), checkpoint(dest)
{
    body.setOrigin(sf::Vector2f(50.0f, 50.0f));
    body.setTexture(*texture);
    body.setScale(scale);
    body.setPosition(position);
}
```

Initialize the destination and checkpoint of class Portal

C++ 11 FEATURES (5) – DEFAULT KEYWORD

- Default keyword was used for defining destructors where no memory was dynamically allocated

```
class Coin : public Item
{
    public:
        Coin(sf::Texture* texture, v2f position);
        ~Coin() = default;
```

Default keyword used for Coin class destructor

C++ 11 FEATURES (6) – OVERRIDE KEYWORD

- Override keyword was used in derived classes of Item to ensure that the function is virtual and is overriding the pure virtual function in class Item

```
virtual void UpdatePlayer(Player &player) = 0;
```

→ **Pure virtual function in class Item**

```
void UpdatePlayer(Player &player) override;
```

→ **Override UpdatePlayer() in derived classes**

```
void Coin::UpdatePlayer(Player &player)
{
    player.setScore(player.getScore() + 100);
    body.setPosition(v2f(-1000.0f, 0.0f));
}
```

C++ 11 FEATURES (7) - USING KEYWORD FOR TYPE ALIAS

- Using keyword used for introducing names used synonymous for certain types

```
using v2u = sf::Vector2u;
using v2f = sf::Vector2f;
```

→ **declare v2u and v2f**

```
v2f GetPosition() { return body.getPosition(); }
v2f GetOrigin() { return body.getOrigin(); }
```

→ **Use v2f instead of sf::Vector2f**

STYLE GUIDELINES (1, 2) – NAMING CONVENTIONS

- Class names should be in UpperCamelCase, meaning that the first letter of every word should be capitalized

```
class Animation
```

- Variables with a small scope can have small names

```
for(auto i = 0u; i < portals.size(); i++)
{
...
}
```

STYLE GUIDELINES (3, 4, 5) – NAMING CONVENTIONS

- Names of classes, functions, variables etc., are appropriate with clear meaning

```
void resizeView(const sf::RenderWindow &window, sf::View &view)
```

- Names should be included in parameters of function declarations

```
void Update(int row, float deltaTime, bool faceLeft);
```

- Global constants should be capitalized with underscore separaters

```
static const float VIEW_HEIGHT{512.0f};
```

STYLE GUIDELINES (6, 7) – CLASSES

- Variables within classes shall always be initialized

```
Animation::Animation(sf::Texture* texture, v2u imageCount, float switchTime) :
imageCount(imageCount), switchTime(switchTime)
{
    totalTime = 0.0f;
    currentImage.x = 0;

    uvRect.width = texture->getSize().x / static_cast<float>(imageCount.x);
    uvRect.height = texture->getSize().y / static_cast<float>(imageCount.y);

}
```

- Indicate in source files whether a method is virtual or not

```
/* virtual */
void Potion::UpdatePlayer(Player &player) {
    player.addLives();
    body.setPosition(v2f(-1000.0f, 0.0f));
}
```

STYLE GUIDELINES (8, 9, 10) – FORMATTING

- Place braces under and inline with keywords

```
if(!statsRetrieved && player.getFinishStatus())
{
   scores.setString(scoresDisplay(player, timeTaken, name));
   scores.setPosition(v2f(14700.0f, -250.0f));
   statsRetrieved = true;
}
```

- Braces without contents shall be put on same line

```
Item::Item() { }
```

- Default case should always be present in switch statements

```
switch (event.type)
{
   ...
   default:
   break;
}
```

STYLE GUIDELINES (11, 12) – FILES

- Header files shall include use of macros that guard files to protect against multiple inclusion

```
#ifndef Player_h
#define Player_h
...
#endif
```

- Inline functions in header files are only defined when they are small (10 lines or fewer)

```
int getScore() { return score; }
int getLives() { return lives; }
bool getKeyStatus() { return hasKey; }
```

OUR THOUGHTS

- What worked
- What didn‘t work
- What we learnt
- What we are proud of
- Worth the time?
- Thoughts on interdependencies within group