1.

Actor(int imageID, double startX, double startY, int dir, double depth, StudentWorld\* world);

* Constructor that uses GraphObject

virtual ~Actor() {};

* Empty destructor as cleanUp() in the student world class handles this

virtual void doSomething() = 0;

* Pure virtual function as not all actors behave the same and has the same doSomething method

virtual void move() = 0;

* Pure virtual function as not all actors move the same

virtual bool crossover();

* True for all actors except and ground actors

virtual void bonk() {};

* Virtual empty bonk method as not all actors do anything when bonked so this is empty

virtual bool isEnemy();

* Virtual function that is true for everything but enemies, thus enemies redefines this as true

StudentWorld\* getWorld() const;

* Returns a pointer to the current world and never has to be redefined

void setStatus();

* Sets player to dead and is never updated so it is not virtual

bool getStatus();

* Returns the current actor status and is never updated by other classes

virtual void damaged();

* setStatus to dead and is virtual as enemies are the only one that behave differently when dead

virtual bool isKoopa()

* False for everything except Koopas so has to be virtual and when Koopas die they must create a shell

Peach Class:

Peach(int imageID, double startX, double startY, int dir, double depth, StudentWorld\* world);

* Inherits the actor class and constructs peach

virtual ~Peach() {};

* An empty virtual destructor for peach as cleanUp() handles this

virtual void doSomething();

* A virtual function that redefines the pure virtual doSomething() function for Peaches unique actions. This checks if she has powerups, handles her jump and falling behaviors, calls move(), and bonks Actors in her way

virtual void move();

* A virtual function that redefines the pure virtual move() function for how Peach moves by getting the arrow key inputted and checking for actors in her path

virtual void bonk();

* A virtual function that redefines bonk for when Peach is bonked and detects whether to decrease her health or not if she is invincible

int getHealth();

* Returns Peach's current health and is only used by Peach object

void decHealth();

* Decreases Peach’s health and is only used by Peach object

void incHealth();

* Increases Peaches health and is only used by Peach object

void setInvin();

* Sets Peach to invincible and is only used by Peach object

void setFlow();

* Sets Peach to having flower power and is only defined by Peach object

void setJum();

* Sets Peach to having the jump power and is only defined by Peach object

bool hasJum();

* Returns if Peach has the jump power right now or not and is only defined by Peach object

bool hasInvin();

* Returns if Peach is invincible right now or not and is only defined by object

bool hasFlow();

* Returns if Peach has a flower power or not and is only defined by Peach object

Ground(int imageID, double startX, double startY, StudentWorld\* world);

* Inherits the actor class and constructs Ground objects

virtual ~Ground() {};

* An empty virtual destructor for Ground as cleanUp() handles this

virtual void doSomething() {};

* Redefines the pure virtual doSomething function

virtual void move() {};

* Ground doesn't move so this is empty and defines the purevirtual method

virtual bool crossover();

* Virtual as redefines to false since ground you cant move over

Block(double startX, double startY, StudentWorld\* world, int goodie);

* Inherits Ground class

virtual ~Block() {};

* An empty virtual destructor for blocks as cleanUp() handles this

virtual void bonk();

* Virtual function that releases goodies when needed to

void setGoodie();

* Only needed by block objects and sets goodie to false

Pipe(double startX, double startY, StudentWorld\* world);

* Inherits Ground class

virtual ~Pipe() {};

* An empty virtual destructor for peach as cleanUp() handles this

Goal(int imageID, double startX, double startY, StudentWorld\* world);

virtual void move() {};

* Goals cant move so this is redefined and virtual

virtual void doSomething() {};

* Goals doSomething does nothing so this redefines it and is virtual

virtual ~Goal() {};

* An empty virtual destructor for goals as cleanUp() handles this

Flag(double startX, double startY, StudentWorld\* world);

* Inherits Goals

virtual ~Flag() {};

* An empty virtual destructor for flags as cleanUp() handles this

virtual void doSomething();

* Redefines doSomething to add points and play end level sound and end the level

Mario(double startX, double startY, StudentWorld\* world);

* Inherits Goals

virtual ~Mario() {};

* An empty virtual destructor for mario as cleanUp() handles this

virtual void doSomething();

* Redefines doSomething to add points and finish the game

Enemies(int imageID, double startX, double startY, StudentWorld\* world);

* Inherits actor class

virtual ~Enemies() {};

* An empty virtual destructor for Enemies as cleanUp() handles this

virtual void move();

* Goombas and Koompas move the same so this refines the pure virtual function is virtual

virtual void doSomething();

* Goombas and Koompas act the same so this bonks peach when needed and redefines the pure virtual function and is virtual

virtual bool isEnemy();

* Returns true and refines the virtual inherited function so it is virtual

virtual void bonk();

* Redefines bonk for all enemies and is virtual since it redefines the pure virtual function

virtual void damaged();

* Increases score and sets status and redefines the inherited version from Actor

Goomba(double startX, double startY, StudentWorld\* world);

* Inherits enemies
* Inherits enemies

virtual ~Goomba() {};

* An empty virtual destructor for Goomba as cleanUp() handles this

Piranha(double startX, double startY, StudentWorld\* world);

* Inherits Enemies

virtual ~Piranha() {};

* An empty virtual destructor for piranha as cleanUp() handles this

virtual void move() {};

* Redefines move as Piranhas dont move and is thus virtual

virtual void doSomething();

* Virtual function that attacks Peach and faces her and is virtual as defines the pure virtual function

Koopa(double startX, double startY, StudentWorld\* world);

* Inherits Enemies

virtual ~Koopa() {};

* An empty virtual destructor for Koopa as cleanUp() handles this

virtual bool isKoopa() {return true;}

* Only defined by Koopa so is a normal function

Projectile(int imageID, double startX, double startY, int startDir, StudentWorld\* world);

virtual ~Projectile() {};

* An empty virtual destructor for Projectiles as cleanUp() handles this

virtual void doSomething();

* Virtual function that is used by Fireball and Shells

virtual void move();

* Virtual function that refines the pure virtual method as Projectiles all move the same way

Shell(double startX, double startY, int startDir, StudentWorld\* world);

* Inherits from Projectile

virtual ~Shell() {};

PFireball(double startX, double startY, int startDir, StudentWorld\* world);

* Inherits from Projectile

virtual ~PFireball() {};

* An empty virtual destructor for PFireball as cleanUp() handles this

Fireball(double startX, double startY, int startDir, StudentWorld\* world);

* Inherits from Projectile

virtual ~Fireball() {};

* An empty virtual destructor for Fireball as cleanUp() handles this

virtual void doSomething();

* Virtual since redefines the function to hit Peach

PowerUp Class:

PowerUp(int imageID, double startX, double startY, StudentWorld\* world);

* Inherits from actor

virtual ~PowerUp() {};

virtual void move();

* All powerups move the same it refines the pure virtual function

void Power();

* Only used by power objects

Flower(double startX, double startY, StudentWorld\* world);

* Inherits from power up and creates a flower object

virtual ~Flower() {};

* An empty virtual destructor for Flower as cleanUp() handles this

virtual void doSomething();

* Redefines the overall pure virtual method for flower and sets lets peach shoot fireballs

Mushroom(double startX, double startY, StudentWorld\* world);

* Inherits from power up and creates a flower object

virtual ~Mushroom() {};

* An empty virtual destructor for Mushrooms as cleanUp() handles this

virtual void doSomething();

* Redefines the pure virtual function and sets the jump powerup on and calls move();

Star(double startX, double startY, StudentWorld\* world);

* Inherits from power up and creates a flower object

virtual ~Star() {};

* An empty virtual destructor for Star as cleanUp() handles this

virtual void doSomething();

* Redefines the pure virtual method and calls move() and sets player to invincible

StudentWorld(string assetPath);

virtual ~StudentWorld();

virtual int init();

virtual int move();

virtual void cleanUp();

void eraseDead();

bool onGround(double x, double y);

* Only defined by StudentWorld and is used to check if there is a ground type object

bool Overlap(double x, double y);

* Only defined by StudentWorld and is used to check for another actor

bool onPeach(double x, double y);

* Only defined by StudentWorld and is used to check is peach is there

bool onEnemy(double x, double y);

* Only defined by StudentWorld is used to check for enemies

void bonkActor(double x, double y);

* Only defined by StudentWorld and bonks the actor at x, y

void bonkPeach();

* Only defined by StudentWorld and bonks Peach

void newActor(Actor\* actor);

* Only defined by StudentWorld and creates new actors

Peach\* getPlayer();

* Only defined by StudentWorld and returns a pointer to peach

void attackEnemy(double x, double y);

* Only defined by StudentWorld and damages the Enemy at x,y

bool isAlive(double x, double y);

* Only defined by StudentWorld and checks if the actor is alive at that coord

void displayText();

* Only defined by StudentWorld and displays the game text using osstreams

void setGame();

* Only defined by StudentWorld and sets the game to true and finishes the game

void setLevel();

* Only defined by StudentWorld and sets the level to true and finishes the level

2. I don’t know of any bugs in my classes or parts of the program that I did not finish.

3. It wasn't specified but for all my Overlap functions I treated entering 1 sprite width and height as overlapping so then the actors never overlap unnecessarily.