Duncan Di Mauro

UID #805163177

2/27/2022

CS32 Project 3 Report – Super Peach Sisters

**1.**

My design of the Actor inheritance hierarchy is:

GraphObject

Actor

Peach

Block

Pipe

LevelEnder

Enemy

Koopa

Goomba

Piranha

Goodie

Flower

Mushroom

Star

Projectile

PiranhaFireball

PeachFireball

Shell

In my StudentWorld.h file, there is only one class, with the hierarchy being:

GameWorld

StudentWorld

**A high-level description of each of my public member functions in each of my classes, and the reason for why I chose to define each member function in its host class is as follows:**

Public functions in the Actor base class:

bool isAlive() const;

This function determines whether an Actor is in the alive state (true) or dead state (false).

void setDead();

This function sets an Actor’s alive state to false (i.e., the Actor is dead).

bool isCollidable() const;

This function determines whether an Actor has the property of being collidable (Actors that are not collidable prevent other actors from moving in a way which would cause a collision between them).

bool isDamagable() const;

This function determines whether an Actor can be damaged (ex. from a fireball).

void reverseDir();

This function reverses the current direction of the actor.

StudentWorld\* getWorld();

This function returns a pointer to the current StudentWorld object being used by all actors.

void doSomethingIfAlive();

This function is called by the StudentWorld class for all actors that have been created. This function calls a pure virtual auxiliary function titled “virtual void doSomething() = 0” which exists in all derived classes. I chose to define a pure virtual version of the doSomething() function in my base Actor class because all actors are able to do something, and each type of actor does something in a different way.

virtual void BonkMe(bool bonkerIsStarPowerPeach);

This function tells an actor that an attempt has been made for it to be bonked, and whether or not the “bonker” is a Peach with the star power currently active. The version of BonkMe() in my base Actor class simply returns and does nothing. I chose to define a virtual version of the BonkMe() function in my base Actor class because all actors can have an attempt made to bonk them – some actors do nothing when this attempt is made, while some do specific things.

virtual void damageMe();

This function tells an actor that an attempt has been made for it to be damaged. The version of damageMe() in my base Actor class simply returns and does nothing. I chose to define a virtual version of the damageMe() function in my base Actor class because all actors can have an attempt made to damage them – some actors do nothing when this attempt is made, while some do specific things.

bool fall(double checkBelow, double moveDown);

All moving actors in this game fall down x\_ amount of pixels if there is no object that blocks movement x\_amount of pixels below them. This function checks if there is an object that blocks movement checkBelow amount of pixels below an actor, and if not, it moves the actor moveDown pixels down and returns true. If not, it returns false.

bool AutoMoveOrTurn(double spacesPerTick);

This function returns true and moves an actor spacesPerTick amount of pixels horizontally in the direction that they are facing if there is no object that blocks movement spacesPerTick pixels in front of them. If there is, it will instead return false and will reverse the actor’s direction.

Public functions in the Peach derived class:

Peach(int pos\_x, int pos\_y, StudentWorld\* sw)

This function creates a Peach actor at location pos\_x, pos\_y in the current StudentWorld object.

void BonkMe(bool bonkerIsStarPowerPeach);

This function causes peach to be Bonked in a certain way, which happens when peach comes in contact with an enemy while she is not invincible due to a star power or a temporary invincibility.

void setHP(int hp);

This function sets Peach’s health points to the value passed into this function.

void gainStarPower(int ticks);

This function causes Peach to gain the star power for a certain amount of ticks, which makes her invincible and kills any enemies when coming into contact with them.

void gainShootPower();

This function causes Peach to gain the shoot power, which lets her shoot fireballs.

void gainJumpPower();

This function causes Peach to gain the jump power, which increases her jump height.

bool hasStarPower() const;

This function determines whether peach currently has the star power.

bool hasShootPower() const;

This function determines whether peach currently has the shoot power.

bool hasJumpPower() const;

This function determines whether peach currently has the jump power.

*All other classes that are derived from the Actor base class do not have any public member functions other than their own constructor. Each constructor serves to initialize an object with the relevant starting parameters such as the StudentWorld it is being sent to, its starting coordinates, its alive state, its starting direction, if it is collidable, if it is damageable, its imageID which is used to create the correct image, and any other private member variables which serve purposes specific to those derived classes.*

Public functions in the StudentWorld class:

virtual int init();

This function initializes the game with all Actors in their proper starting positions and conditions.

virtual int move();

This function causes each Actor to doSomething for every tick of the game.

virtual void cleanUp();

This function removes all Actors from the current StudentWorld

void LevelCompleted();

This function notifies StudentWorld that the current level has been completed.

void GameIsWon();

This function notifies StudentWorld that the game has been won.

void FacePeach(Actor\* self);

This function sets the current actor’s direction so that they face peach.

double dXfromPeach(Actor\* self);

This function returns the difference in x-coordinates from Peach to the current actor.

double dYfromPeach(Actor\* self);

This function returns the difference in y-coordinates from Peach to the current actor.

bool overlap(double x, double y, Actor\* self);

This function checks if the current actor will overlap with any other actors if it moves to location x, y.

bool overlapsWithPeach(Actor\* self);

This function checks if an Actor overlaps with Peach.

bool bonkOverlappingPeach(Actor\* bonker);

This function checks if Peach overlaps with the current actor, and bonks peach and returns true if it does.

void BonkOverlappingActors(Actor\* bonker);

This function bonks any Actors that currently overlap with the Actor passed into this function.

bool DamageOverlappingActor(Actor\* damager, bool damagedByPeach);

Depending on whether or not Peach is the actor inflicting damage, this function damages any actors that currently overlap with the Actor damager.

bool spaceBlocked(double x, double y, Actor\* self);

This function determines whether an Actor is able to move to location x, y, or if that space is blocked.

bool moveOrBonk(int x, int y, Actor\* self);

This function moves an Actor to a location if possible (and returns true). If it is blocked by an object, it will bonk that object instead.

void addActor(Actor\* a);

This function creates a new actor in the current StudentWorld.

void setPeachHP(int hp) const;

This function sets Peach’s hp to a certain value.

void grantStarPower() const;

This function grants Peach the star power.

void grantShootPower() const;

This function grants Peach the shoot power.

void grantJumpPower() const;

This function grants Peach the jump power.

**2. A list of all functionality that you failed to finish as well as known bugs in your classes.**

I did not fail to finish any functionality and there are not any known bugs in my classes.

**3. A list of other design decisions and assumptions you made; e.g., “It was not specified what to do in situation X, so this is what I decided to do.”**

In the demo executable, when Peach ran into a pipe, there was a “bonk” noise. However, the specification said that a pipe should do nothing when bonked. I decided to follow the specification.

In the demo executable, when Koopas or Goombas either need to turn around due to a space being blocked or due to them not being allowed to walk over a ledge, they do so one pixel late if reversing their direction to face right, and one pixel earlier when reversing their direction to face left (so if they run into a pipe or wall on their left side, one of their pixels will collide with the pipe. However, if they run into a pipe or wall on the right side, they will turn around while they are 2 pixels away from the wall or pipe). In my code, I made it so this does not happen and made it so that these enemies never collide with a wall or pipe, and will always turn around while 1 pixel away from a wall, pipe, or opening in the floor.

In the demo executable, the text at the top of the screen has two spaces between the number describing lives and the world “Level.” It also has two spaces between the number describing level and the word “Points.” In the specification, there is only one space. In this case, I followed the demo executable.

In the specifications, it said that we could choose whether a new actor does something during the tick that they are created or afterwards. I decided to make my new actors do something during the tick after they are created.