# Project 3 Report

### Actor.h/Actor.cpp

## Actor Base Class

**virtual void doSomething() = 0;**

This function was done as a pure virtual function because I need every single actor to return something since I call this function when going through my vector.

**virtual bool solidObject() = 0;**

This function was done as a pure virtual function because I have derived classes that must specify whether an object is “overlapable” or not.

**StudentWorld\* getWorld(){return m\_world;}**

This function helps other classes obtain public member functions from the game’s world.

**bool checkifAlive(){ return isAlive;}**

This function can be used by any of the derived classes as the functionality should be the same for all derived classes.

**void die(){ isAlive = false;}**

This function can be used by any of the derived classes as the functionality should be the same for all derived classes.

**virtual bool isEnemy(){return false;}**

This function can be used by any of the derived classes that are not enemies and can be redefined for actors who are enemies. This is for identification.

**virtual bool isDamageable(){return false;}**

This function can be used by any of the derived classes that are not damageable and can be redefined for actors who are. This is for identification.

**virtual void damage(){return;}**

This function can be used by any of the derived classes that do nothing when damaged and can be redefined for actors who are damaged.

**virtual void bonk(Actor\* bonker) {return;}**

This function can be used by any of the derived classes that do nothing when bonked and can be redefined for actors who are bonked.

## Pipe and Block Classes – Derived from Actor

**virtual void doSomething(){return;}**

This function is included as it is a pure virtual in the base class and it must be defined. Both pipe and block do nothing.

**virtual void solidObject(){return true;}**

This function is included as it is a pure virtual in the base class and it must be defined. Both pipe and block are solid objects.

## Overlapable Class – Derived from Actor

**virtual void doSomething(){return;}**

This function is included as it is a pure virtual in the base class and it must be defined. In an event where a derived does not call its own doSomething, it will return for the base class’s.

**virtual void solidObject(){return false;}**

This function is included as it is a pure virtual in the base class and it must be defined. Overlapables are not solid objects.

## End Level Class – Derived from Overlapable

**virtual void doSomething();**

This function is included as it is a pure virtual in the base classes. This will be specified for a flag or Mario in each level and does something for each tick.

## Goodie Class – Derived from Overlapable

**virtual void doSomething(){return;}**

This function is included as it is a pure virtual in the base class and it must be defined. In an event where a derived does not call its own doSomething, it will return for the base class’s.

## Flower Class – Derived from Goodie

**virtual void doSomething();**

This function is included as it is a pure virtual in the base classes. This will be specified for this goodie level and does something for each tick.

## Star Class – Derived from Goodie

**virtual void doSomething();**

This function is included as it is a pure virtual in the base classes. This will be specified for this goodie level and does something for each tick.

## Mushroom Class – Derived from Goodie

**virtual void doSomething();**

This function is included as it is a pure virtual in the base classes. This will be specified for this goodie level and does something for each tick.

## Projectile Class – Derived from Overlapable

**virtual void doSomething(){return;}**

This function is included as it is a pure virtual in the base class and it must be defined. In an event where a derived does not call its own doSomething, it will return for the base class’s.

## Shell Class – Derived from Projectile

**virtual void doSomething();**

This function is included as it is a pure virtual in the base classes. This will be specified for this goodie and does something for each tick.

## Piranha Fireball Class – Derived from Projectile

**virtual void doSomething();**

This function is included as it is a pure virtual in the base classes. This will be specified for this goodie and does something for each tick.

## Peach Fireball Class – Derived from Projectile

**virtual void doSomething();**

This function is included as it is a pure virtual in the base classes. This will be specified for this goodie and does something for each tick.

## Enemy Class – Derived from Overlapable

**virtual void doSomething(){return;}**

This function is included as it is a pure virtual in the base class and it must be defined. In an event where a derived does not call its own doSomething, it will return for the base class’s.

**virtual bool isEnemy(){return true;}**

This function can be used to identify that any derived of the enemy class is an enemy. This is for identification.

**virtual bool isDamageable(){return true;}**

This function can be used to identify that any derived of the enemy class can be damaged. This is included so those of the derived will not call Actor’s isDamageable which says it is not an enemy. This is used for identification.

## Goomba Class – Derived from Enemy

**virtual void doSomething();**

This function is included as it is a pure virtual in the base classes. This will be specified for this enemy and does something for each tick.

**virtual void bonk(Actor\* bonker) ;**

This function is included as it does not simply return when bonked as the base class’s bonk() does. This will be specified for this enemy and does something for each tick.

## Koopa Class – Derived from Enemy

**virtual void doSomething();**

This function is included as it is a pure virtual in the base classes. This will be specified for this enemy and does something for each tick.

**virtual void bonk(Actor\* bonker) ;**

This function is included as it does not simply return when bonked as the base class’s bonk() does. This will be specified for this enemy and does something for each tick.

## Piranha Class – Derived from Enemy

**virtual void doSomething();**

This function is included as it is a pure virtual in the base classes. This will be specified for this enemy and does something for each tick.

**virtual void bonk(Actor\* bonker) ;**

This function is included as it does not simply return when bonked as the base class’s bonk() does. This will be specified for this enemy and does something for each tick.

## Peach Class – Derived from Actor

**virtual void doSomething();**

This function is included as it is a pure virtual in the base classes. This will be specified for Peach and does something for each tick.

**virtual void bonk(Actor\* bonker) ;**

This function is included as it does not simply return when bonked as the base class’s bonk() does. This will be specified for Peach and does something for each tick.

**void addHP(int hp);**

This is a function used for peach only. Here, I can use it to decrement or increment her hit points.

**void setHP(int);**

This is a function used for peach only. This is a function used for setting the hit points to a specific number.

**void gainInvincibility(int ticks);**

This is a function used for peach only. This is a function used for setting i\_ticks to ticks.

**void gainShootPower();**

This is a function used for peach only. This is a function used for p\_shoot to true.

**void gainJumpPower();**

This is a function used for peach only. This is a function used for p\_jump to true.

**bool isInvincible() const{return p\_invincible;}**

This is a function used for peach only. This is a function used to check whether p\_invincible is true/false.

**bool hasShootPower() const{return p\_shoot;}**

This is a function used for peach only. This is a function used to check whether p\_shoot is true/false.

**bool hasJumpPower() const{return p\_jump;}**

This is a function used for peach only. This is a function used to check whether p\_jump is true/false.

### StudentWorld.h/ StudentWorld.cpp

## Student World Class

**bool addFlower(int x, int y) const;**

**bool addStar(int x, int y) const ;**

**bool addMushroom(int x, int y) const;**

These add in the goodies for when Peach bonks a goodie block.

**// projectiles**

**bool addShell(int x, int y, int dir);**

**void addPeachFireball(int x, int y, int dir);**

**void addPiranhaFireball(int x, int y, int dir);**

These add in the projectiles into the world when they are needed.

**// Check Position**

**bool checkPos(int x, int y, Actor\* act) const;**

Helps check the position of the given coordinates and actor.

**// Bonk Peach**

**void bonkPeach(Actor\* bonker) const;**

This function bonks Peach.

**// If Peach overlaps, bonk the second actor**

**bool overlapThenBonk(Actor\* a1) const;**

**// If Peach overlaps bonker, bonk 'er and return true; otherwise,**

**// return false.**

**bool bonkOverlappingPeach(Actor\* bonker) const;**

**// If Peach overlaps damager, damage her and return true; otherwise,**

**// return false.**

**bool damageOverlappingPeach(Actor\* damager) const;**

**// If a non-Peach actor overlaps damager, damage that non-Peach actor**

**// and return true; otherwise, return false.**

**bool damageOverlappingActor(Actor\* damager) const;**

**// Return true if a overlaps Peach; otherwise, return false.**

**bool overlapsPeach(Actor\* a) const ;**

**// find out how far actor is from peach**

**int getPeachTargetingInfo(Actor\* a) const;**

**// Set Peach's hit points to hp.**

**void setPeachHP(int hp) const;**

**// Grant Peach invincibility for this number of ticks.**

**void grantInvincibility(int ticks) const;**

**// Grant Peach Shoot Power.**

**void grantShootPower() const;**

**// Grant Peach Jump Power.**

**void grantJumpPower() const;**

**// Return true if a overlaps an enemy; otherwise, return false.**

**bool overlapWithEnemy(Actor\* primary) const;**

**// if blocking position, bonk the second actor**

**bool blockThenBonk(int x, int y, Actor\* a1, bool bonk) const;**

**// if blocking position, set true**

**bool isBlocked(int x, int y, Actor\* a1) const;**

**// Return main character**

**Peach\* getPeach() const;**

**// is this spot empty?**

**bool isEmpty(double x, double y) const;**

**// overlap**

**bool overlap(int x1, int y1, int x2, int y2) const;**

**// Signify we have started the game**

**void startTheGame();**

**// Signify we have finished this level**

**void finishedLevel(bool done);**

**// Signify we have won the game**

**void wongame(bool done);**

**// Mario is here**

**bool marioinGame(){return marioisHere;}**

### Bugs/Other Design Decisions

Some bugs I did not have sufficient time to fix are:

1. Goombas in the second level move in mid-air when there is no block underneath it. I intend to fix it by including stricter criteria for when it is okay for the object to move.
2. Also, I am not sure if the way I implemented the temporary invisibility works as intended. I made it so even when I have star power, I reset those ticks to 10 for temporary (when I need to).