**1.**

**Actor class: ( derived from GraphObject)**

We have a constructor and a destructor. Then It have a not virtual getworld function that return a private StudentWorld pointer member m\_world since each kind of actor may need to interact with other actors thus they all need a way to made that influence through the studentworld.

It has a pure virtual doSomething function since each actor do something in their own way.

It has a pure virtual blockingOrNot function that return a bool that indicate whether this actor blocking others movement or not. It has a pure virtual damagableOrNot function that return a bool that indicate whether this actor is damageable or not. These are pure virtual since each kind of actor may different about their blocking-or-not property and damageable-or-not property.

**It has a isOverlap function that take two double parameter(x,y) that return a bool to** indicate whether this actor is taking up this position. This is nor virtual since all actor have their x,y position stored and we can examine this in the same way.

It has a pure virtual bonk function since each actor react to bonk in different ways.

It has a non-virtual setdeath function that set the actor’s aliveState to false, indicating death. It has a non-virtual aliveOrNot function that return a Boolean indicating the aliveState of the actor. Since the alive property is shared by all kinds of actor, these two functions are just non\_virtual;

It has a virtual getDamage function that setdeath the actor when the actor gets damage. Since koopa and peach have different behavior (more than just simply setdeath), they will override this function thus makes this function virtual.

**Peach class: (derived from actor)**

We have constructor for peach, its own doSomething function and bonk function which are not virtual since they just follow peach’s behavior.

It have a blockingOrNot function that return false since peach doesn’t block other actors and a damagableOrNot function that return true since peach is damageable by other actors. These are both non-virtual that override actor’s corresponding functions.

It has a power function which have two parameter to indicate the type of the power and their open or close states. It will serve to empower of disable peach’s certain power. It has a checkPower function which have a parameter to indicate which kind of power we are looking at, and returns a bool indicate the peach have that power or not. It has a hitpoint2 function which simply set peach’s hit point to two. These functions are all non-virtual since they just serve peach’s behavior.

It overrides the actor’s getDamage function, which make peach behave like being bonked when getting damage.

**Blocking\_objects class: (derived from actor)**

We have constructor for blocking\_objects, its own doSomething function which actually does nothing since blocking objects really do nothing.

It has a bonk function that release object’s corresponding holding goodies.

It has a blockingOrNot function that return true since blocking objects do block other actors and a damagableOrNot function that return false since blocks and pipes are not damageable by other actors. These are both non-virtual that override actor’s corresponding functions.

**Block class: (derived from Blocking\_objects)**

We have a constructor for block that construct itself and set its goodies holding state.

**Pipe class: (derived from Blocking\_objects)**

We have a constructor for pipe that construct itself and set its goodies holding state to 0(null).

**Portal class: (derived from actor)**

We have constructor for portal and its pure virtual doSomething function since Mario and flag behave differently In their doSomething.

It has a blockingOrNot function that return false since portal objects don’t block other actors and a damagableOrNot function that return false since mario and flags are not damageable by other actors. These are both non-virtual that override actor’s corresponding functions.

It has a bonk function that does nothing since portal cannot be bonked. So it does nothing when bonked.

**Flag class: (derived from portal)**

It has a constructor and a doSomething function that take peach to the next level, this is flag’s own behavior, so I make it non-virtual.

**Mario class: (derived from portal)**

It has a constructor and a doSomething function finish the game, this is mario’s own behavior, so I make it non-virtual.

**Goodies class(derived from actor)**

It has a constructor that construct itself by different type of goodies.

It has a virtual doSomething function that implemented some common bahavior like moving of the goodies. Since each goodies have their own other behavior, this function is virtual.

It has a blockingOrNot function that return false since goodies objects don’t block other actors and a damagableOrNot function that return false since star, mushroom and flower are not damageable by other actors. These are both non-virtual that override actor’s corresponding functions.

It has a bonk function that does nothing since portal actually cannot be bonked. So it does nothing when bonked.

**Mushroom class(derived from goodies)**

It has a constructor and a doSomething function that give peach jump power and then call the goodies doSomething for moving. These are non-virtual since this are just belong to mushroom.

**star class(derived from goodies)**

It has a constructor and a doSomething function that give peach invincibility power and then call the goodies doSomething for moving. These are non-virtual since this are just belong to mushroom.

**flower class(derived from goodies)**

It has a constructor and a doSomething function that give peach fireball power and then call the goodies doSomething for moving. These are non-virtual since this are just belong to mushroom.

**Fireball class(derived from actor)**

It has a constructor that construct proper fireball objects by the type parameter.

It has a virtual doSomething function that implemented some common bahavior like moving of the fireball. Since each kind of fireballs have their own other behavior, this function is virtual.

It has a blockingOrNot function that return false since fireball objects don’t block other actors and a damagableOrNot function that return false since peach\_fireball, piranha\_fireball and shell are not damageable by other actors. These are both non-virtual that override actor’s corresponding functions.

It has a bonk function that does nothing since portal actually cannot be bonked. So it does nothing when bonked.

**Piranha\_fireball class(derived from fireball)**

It has a constructor and a dosomething function that set death peach if it is overlap with peach, then call fireball’s doSomething function for movement.

**Peach\_object class(derived from fireball)**

It has a constructor and a dosomething function that set death peach if it is overlap with damageable object other than peach, then call fireball’s doSomething function for movement.

**Shell class(derived from Peach\_obejct)**

It just has a constructor to properly construct itself.

**Peach\_ fireball(derived from Peach\_obejct)**

It just has a constructor to properly construct itself.

**Monster class(derived from actor)**

It has a constructor that construct different monster object properly according to their type.

It has a virtual doSomething function which are implemented to attack peach and move, which are the behavior shared by all monsters. Since each kind of monsters also have each’s special behavior, doSomething function is virtual to be overridden.

It has a blockingOrNot function that return false since monster objects don’t block other actors and a damagableOrNot function that return true since goomba, piranha and koopa are damageable by other actors. These are both non-virtual that override actor’s corresponding functions.

It has a virtual bonk function that check whether peach have star power. If does, then play sound, increase the score and set death to this monster object. Since koopa behave differently when bonked, this function is virtual to be overridden.

**Goomba class(derived from monster)**

It has a constructor that properly construct itself.

It has a doSomething function that check its aliveState. If dead, do nonthing. If alive, then call monster’s doSomething function to move.

**Piranha class(derived from monster)**

It has a constructor that properly construct itself.

It has a doSomething function that turn its direction, check peach position to fire piranha fire ball. This is non-virtual since it is piranha’s own behavior.

**koopa class(derived from monster)**

It has a constructor that properly construct itself.

It has a doSomething function that check its aliveState. If dead, do nonthing. If alive, then call monster’s doSomething function to move.

It override monster’s bonk function by also release a shell when setDeath to itself, it is non-virtual since its koopa’s own behavior.

It override actor’s getDamage by release a shell before setDeath, it is non-virtual since its koopa’s own behavior.

2. no functionality that I failed to finish as well as known bugs

3.no design decisions and assumptions I made.