**Report**

1.A high-level description of each of your public member functions in each of your classes

why you chose to define each member function in its host class

why (or why not) you decided to make each function virtual or pure virtual.

I firstly designed a general base class called Actor, and it is inherited from the class GraphObject (which is already been given from the instruction).

I make the function doSomething(), bonk(), and damage() to be pure virtual because every objects derived from the Actor class will doSomething in every tick, and each class may be able to be bonked and damaged. Each of them behaves differently.

I then make the Actor class member functions checkAlive()(which check whether the actor is alive), setPrevent()(which set the actor the ability to block other actors), checkPrevent()(which check wether the actor can block other actors), setDead()(which set the actor to be non-alive), checkDamagable()(which check whether the actor can be demaged), setdamagable()(which set the actor the ability to be damaged by other actors) since every objects derived from the Actor class will need to use those functions during sometimes in execution and each classes uses them in the same way.

I then designed three derived classes directly derived from the base class Actor: class Creature, class Buff, class Building.

For the Creature class, I designed it as an abstract class (since it still have the pure virtual function doSomething(), bonk(), and damage() derived from the base class Actor). It has more member function: blood() which can return the creature’s health point; the loseblood(int num) which help decrease the creature’s health point; the setblood2() which help set the creature’s blood to 2; the badmove() function which illustrate the common steps that the creatures may use during their execution, such as checking whether there are blocks under the creature that permits it to move forward; and the HP() which returns the health point of the creature;

For the Peach class that derived from the Creature class, I designed it as a derived non-abstract class. Here, all the functions within the Peach class are non-pure virtual function. The virtual function doSomething() will specify what the Peach will do during each tick(and thus it will perfrom different from other classes and is non-pure virtual); the setShootPower() will set the ShootPower of Peach to be true; the noShootPower() will set the ShootPower of Peach to be false; the checkShootPower() will check whether the peach have the ShootPower or not; the setStarPower(int a) will specify the StarPower amount by entering an int a; the noStarPower() will set the StarPower to be 0; the checkStarPower() will check whether the Peach have StarPower or not; the setJumpPower() will set the JumpPower of Peach to be true; the noJumpPower() will set the JumpPower of Peach to be false; the checkJumpPower() will check whether the peach has the JumpPower or not; the setInvincible() will set the invincible ability of peach to be true; the setInvincibleT(int a) will help set the time that Peach will have the invincible to be a; the setnotInvincible() will set that Peach do not have the invincible ability; the checkInvincible() will check whether the peach has the ability to be invincible; the settempInvincible() will check whether the peach has the temporary invincibility; the settempInvincibleT(int a) will set the time of being temporary invincibility to be a; setnottempInvincible() will disable the peach to have the temporary invincible ability; setCanShoot() will set the peach to have the ability to shoot; the setJump() function can give the peach the ability to jump; the noCanShoot() function will set the peach doesn’t have the ability to shoot; the bonk() function defines how the peach would act if it is bonked by other actors (which is do nothing); the damage() function defines how the peach is damaged by other actors( which is the same as being bonked);

For the Goomba class that derived from the Creature class, I designed it as a derived non-abstract class. Here, all the functions within the Goomba class are non-pure virtual function. The doSomething() defines what the Goomba will do during each tick; the bonk() will defines what the Goomba performs if it is bonked by other actors;

For the Koopa class that derived from the Creature class, I designed it as a derived non-abstract class. Here, all the functions within the Koopa class are non-pure virtual function. The doSomething() defines what the Koopa will do during each tick; the bonk() will defines what the Koopa performs if it is bonked by other actors;

For the Piranha class that derived from the Creature class, I designed it as a derived non-abstract class. Here, all the functions within the Piranha class are non-pure virtual function. The doSomething() defines what the Piranha will do during each tick; the decreaseFire() will decrease the fireDelay time by one; the setFiredelay(int i) will define the fireDelay time as i; the the bonk() will defines what the Piranha performs if it is bonked by other actors; the damage() defines how the actor behaves if it is damaged by other actors;

For the Buff class that derived from the Creature class, I designed it as a derived abstract class. Here, some of the functions within the Buff class are still pure virtual function (such as doSomething(), damage(), bonk()). However, it also have some virtual functon such as refreshPeach() (which defines how balls would move under the spec requirement) and refreshbad() (which defines how the goodie would move under the spec requirement).

For the Flower class that derived from the Buff class, I designed it as a derived abstract class. Here, all the functions within the Flower class are non-pure virtual function. The doSomething() defines what the Flower will do during each tick; the bonk() will defines what the Flower performs if it is bonked by other actors; the damage() function defines what the Flower would be damaged by calling damaged by other actors;

For the Mushroom class that derived from the Buff class, I designed it as a derived abstract class. Here, all the functions within the Mushroom class are non-pure virtual function. The doSomething() defines what the Mushroom will do during each tick; the bonk() will defines what the Mushroom performs if it is bonked by other actors; the damage() function defines what the Mushroom would be damaged by calling damaged by other actors;

For the Star class that derived from the Buff class, I designed it as a derived abstract class. Here, all the functions within the Star class are non-pure virtual function. The doSomething() defines what the Star will do during each tick; the bonk() will defines what the Star performs if it is bonked by other actors; the damage() function defines what the Star would be damaged by calling damaged by other actors;

For the PiranhaFireball class that derived from the Buff class, I designed it as a derived abstract class. Here, all the functions within the PiranhaFireball class are non-pure virtual function. The doSomething() defines what the PiranhaFireball will do during each tick; the bonk() will defines what the PiranhaFireball performs if it is bonked by other actors; the damage() function defines what the PiranhaFireball would be damaged by calling damaged by other actors;

For the PeachFireball class that derived from the Buff class, I designed it as a derived abstract class. Here, all the functions within the PeachFireball class are non-pure virtual function. The doSomething() defines what the PeachFireball will do during each tick; the bonk() will defines what the PeachFireball performs if it is bonked by other actors; the damage() function defines what the PeachFireball would be damaged by calling damaged by other actors;

For the Shell class that derived from the Buff class, I designed it as a derived abstract class. Here, all the functions within the Shell class are non-pure virtual function. The doSomething() defines what the Shell will do during each tick; the bonk() will defines what the Shell performs if it is bonked by other actors; the damage() function defines what the Shell would be damaged by calling damaged by other actors;

For the Flag class that derived from the Actor class, I designed it as a derived abstract class. Here, all the functions within the Flag class are non-pure virtual function. The doSomething() defines what the Flag will do during each tick; the bonk() will defines what the Flag performs if it is bonked by other actors; the damage() function defines what the Flag would be damaged by calling damaged by other actors;

For the Mario class that derived from the Actor class, I designed it as a derived abstract class. Here, all the functions within the Mario class are non-pure virtual function. The doSomething() defines what the Mario will do during each tick; the bonk() will defines what the Mario performs if it is bonked by other actors; the damage() function defines what the Mario would be damaged by calling damaged by other actors;

2. A list of all functionality that you failed to finish as well as known bugs in your classes, e.g. “I didn’t implement the Flower class.” or “My koopa doesn’t work correctly yet so I treat it like a goomba right now.”

According to my own test, none of my function failed.

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.”

It was not specified in what arrange the peach would get the mushroom or the star as its super power, so I decided to set the range of overlapping with the peach with the range greater or smaller than 3 pixels of Peach’s both x direction and y direction.