**Question 1**

**Hierarchy:**

* Actor
  + Goodie
    - Flower
    - Mushroom
    - Star
  + Peach
  + Block
  + Flag
  + Projectile
    - PeachProjectile
    - PiranhaFireball
  + Enemy
    - MovingEnemy
      * Goomba
      * Koopa
    - Piranha

**Actor:**

**pure virtual void doSomething():**

* This handles what actors do during each tick of the main loop of the game. This is pure virtual as the base actor shouldn’t be able to do anything. This is defined in the base actor class as all actors will have to be able to do something.

**virtual void bonk(actor):**

* This handles what happens to an actor when it is bonked. This is virtual and by default it does nothing as most actors will do nothing when they are bonked, but those who do in fact do something will need to override the function. This is defined in the actor base class as all actors need to be able to be bonked.

**virtual void damage():**

* This handles what happens to an actor when it is damaged. This is virtual and by default it does nothing as most actors will do nothing when they are damaged, but those who do in fact do something will need to override the function. This is defined in the actor base class as all actors need to be able to be damaged.

**virtual bool isSolid():**

* This informs the user whether or not the actor is a solid object that can’t be overlapped with. This returns false by default, but it is virtual as actors that are in fact solid will need to override the function to return true. This is defined in the actor base class as all actors will need to be able to indicate whether or not they are solid.

**virtual bool isDamageable():**

* This informs the user whether or not the actor can be damaged This returns false by default, but it is virtual as actors that are in fact damageable will need to override the function to return true. This is defined in the actor base class as all actors will need to be able to indicate whether or not they can be damaged

**bool isAlive():**

* This informs the user whether or not the actor is alive. This is not virtual as all actors handle their live status in the same way. This is defined in the actor base class as all actors will need to be able to indicate whether or not they are alive.

**StuentWorld getWorld():**

* This returns a reference to the student world. This is not virtual as all actors store the student world in the same way. This is defined in the actor base class as all actors will need access to the student world.

**bool isCollidingWith(x, y):**

* This checks if the bounding box of the actor collides with another bounding box starting at x, y. This is not virtual as all actors have the same bounding boxes and should handle collision detection in the same way. This is defined in the actor base class as all actors will need to be able to check if they collide with other actors.

**bool isCollidingWith(actor):**

* This checks if the bounding box of the actor collides with another actor’s bounding box. This simply calls the implementation of isCollidingWith above.

**void kill():**

* Sets the current life state of the actor to death. This is not virtual as all actors handle their life state in the same way. This is defined in the actor base class as all actors will need to die at some point.

**void relativeMove(dx, dy):**

* This moves the actor relative to its current position. This is not virtual as all actors move in the same way. This is defined in the actor base class as all actors move in the same way.

**Goodie** **(Actor)**

**pure virtual GoodieType getType()**

* This returns the type of the goodie. This is pure virtual as all goodies have a type, but there is no default type. This is in the Goodie base class as all goodies will need to be able to say what type they are.

**pure virtual int getPoints()**

* This returns the number of points earned from the goodie. This is pure virtual as all goodies have a point value, but there is no default point value. This is in the Goodie base class as all goodies will need to be able to say how many points they give out.

**virtual void doSomething():**

* Implements the doSomething method declared in the actor base class. This mainly handles movement of the goodie.

**Flower (Goodie):**

**virtual GoodieType getType()**:

* Implements the getType method declared in the goodie base class.

**virtual int getPoints()**:

* Implements the getPoints method declared in the goodie base class.

**virtual void doSomething():**

* Adds on to the doSomething method declared in the goodie base class.

**Mushroom (Goodie):**

**virtual GoodieType getType()**:

* Implements the getType method declared in the goodie base class.

**virtual int getPoints()**:

* Implements the getPoints method declared in the goodie base class.

**virtual void doSomething():**

* Adds on to the doSomething method declared in the goodie base class.

**Star (Goodie):**

**virtual GoodieType getType()**:

* Implements the getType method declared in the goodie base class.

**virtual int getPoints()**:

* Implements the getPoints method declared in the goodie base class.

**virtual void doSomething():**

* Adds on to the doSomething method declared in the goodie base class.

**Peach (Actor)**

**virtual void doSomething()**

* Implements the doSomething method declared in the actor base class. Handles keypresses, movement, jumping, and powerups.

**virtual void bonk(actor)**

* Implements the bonk method declared in the actor base class. Decreases peaches health and strips her powers if she isn’t invincible. Kills her if she has no more health.

**virtual void damage**

* Implements the bonk method declared in the actor base class. Calls bonk().

**bool isInvincible()**

* This indicates to the user whether or not peach is currently invincible. This is not virtual as peach is the only one who can be invincible.

**void setHP(hp)**

* This sets peaches HP. This is not virtual as peach is the only one who has HP.

**void givePower(goodie)**

* This informs Peach that she now has the provided power. This is not virtual as peach is the only one who has powers.

**bool hasShoot()**

* This informs the user if peach has the shoot power. This is not virtual as peach is the only one who has powers.

**bool hasJump()**

* This informs the user if peach has the jump power. This is not virtual as peach is the only one who has powers.

**bool hasStar()**

* This informs the user if peach has the star power. This is not virtual as peach is the only one who has powers.

**Block (Actor)**

**virtual void doSomething():**

* Implements the doSomething method declared in the actor base class. Does nothing.

**virtual void bonk(actor):**

* Implements the bonk method declared in the actor base class. Handles the releasing of goodies if necessary.

**virtual void isSolid()**:

* Override the isSolid method declared in the actor base class to return true. Blocks are solid objects that can't be overlapped with.

**Flag (Actor):**

**virtual void doSomething():**

* Implements the doSomething method declared in the actor base class. Informs the student world that the level/game is complete if peach overlaps with it.

**Projectile (Actor):**

**virtual void doSomething()**

* Implements the doSomething method declared in the actor base class. Handles movement of the projectile and death on collision.

**PeachProjectile (Projectile)**

**virtual void doSomething():**

* Adds on to the doSomething method implemented in the projectile base class. Handles damaging anything in its path.

**PiranhaFireball (Projectile)**

* Adds on to the doSomething method implemented in the projectile base class. Handles damaging of peach if colliding with her.

**Enemy (Actor):**

**virtual void doSomething():**

* Implements the doSomething method declared in the actor base class. Bonks peach if colliding with her.

**virtual void bonk(actor):**

* Implements the bonk method declared in the actor base class. Damages the enemy if peach has a star and is the one bonking.

**virtual void damage():**

* Implements the damage method declared in the actor base class. Increases the player's score and kills self.

**virtual void isDamageable():**

* Overrides the isDamageable method declared in the actor base class to return true. Enemies are damageable and need to indicate this.

**MovingEnemy (Enemy):**

**virtual void doSomething():**

* Adds on to the doSomething method declared in the enemy base class. Handles movement of the enemy.

**Goomba (MovingEnemy):**

* Is identical to a MovingEnemy with no additions

**Koopa (MovingEnemy):**

**virtual void bonk():**

* Adds on to the bonk method implemented in the MovingEnemy base class. Releases a shell when killed.

**virtual void damage():**

* Adds on to the damage method implemented in the MovingEnemy base class. Releases a shell when killed.

**Piranha (Enemy):**

**virtual void doSomething():**

* Adds on to the doSomething method declared in the enemy base class. Handles the tracking of peach and shooting of fireballs.

**Question 2:**

* None

**Question 3:**

* It didn’t state whether or not you could check if an actor was peach in the student world, but I found it necessary. I wasn’t sure exactly how to handle collisions, so I made a function that returns nullptr when no actor is found and an actor if one is found. It felt unnecessary to create a separate class for flags/mario, blocks/pipes, and peach fireballs/shells so I set the image id of those using an argument in the respective constructors and a ternary operator in the actor constructor.