1. Classes
   1. StudentWorld
      1. StudentWorld(string assetDir): creates the StudentWorld and sets remainingAliensToKill to however many aliens the player needs to kill on the first level to advance and number\_aliens to 0 because there are no aliens on screen yet
      2. ~StudentWorld(): calls cleanUp() and destructs Student World
      3. Init(): Creates NachenBlaster object and sets m\_player pointer to it, then creates 30 Star objects and pushes it into the m\_actors vector of Actor pointers
      4. Move(): updates Status text, prompts every object in the game to do Something, adds new objects, and removes dead objects.
      5. cleanUp(): removes all objects and resets remainingAliensToKill and number\_aliens
      6. removeDeadObjects(): iterates through m\_actors vector and deletes all objects that are dead
      7. killAlien(): denotes m\_player killing an Alien
      8. subtractAlien(): denotes when an Alien goes offscrean
      9. toAddNewAlien(): determines whether or not to add another Alien and if so, determines what Alien to add
      10. maxAliensOnScreen(): returns max amount of aliens that can be onscreen
      11. getRemainingAliensToKill(): returns remainingAliensToKill
      12. updateStatusLine(): prints status line
      13. addObject(): adds a specified object to the m\_actors vector
      14. canCollide(): returns whether or not an Alien or NachenBlaster can collide with another specified object
      15. checkCollision(): checks if two objects collide and set the collided object to dead and return the collided object’s imageID
      16. getPlayer(): return m\_player pointer
   2. Actor
      1. Actor(): creates an Actor object and sets it m\_id, m\_world, and m\_status
      2. ~Actor(): virtual destructor
      3. getWorld(): returns m\_world pointer
      4. isAlive(): return m\_status
      5. getImageID(): returns m\_id
      6. setDead(): sets m\_status to false
      7. doSomething() and move() are pure virtual because all Actors can do something and move, but in different ways
   3. Star
      1. Star(): creates a star object
      2. ~Star(): virtual destructor
      3. doSomething(): calls move()
      4. move(): move one pixel to the left, if move off screen, call setDead()
   4. Ship
      1. Ship(): sets m\_health
      2. ~Ship(): virtual destructor
      3. getHealth(): return m\_health
      4. decreaseHealth(): subtract m\_health by specificed amount
      5. increaseHealth(): add m\_health by specified amount or set to 50
      6. doSomething() and move() are pure virtual because all Ships can do something and move, but in different ways. sufferDamage() is also pure virtual because all ships take damage, but from different objects
   5. NachenBlaster
      1. NachenBlaster(): sets m\_cabbagePoints andm\_ torpedoes
      2. ~NachenBlaster(): virtual destructor
      3. doSomething(): prompt user for input, check for collisions, and increase m\_CabbagePoints
      4. move(): determines player action, can move in four direction, shoot cabbage or torpedoes, or nothing
      5. sufferDamage(): decrease health based on specified collisionID, if health is 0 or lower, call setDead()
      6. getTorpedoes(): return m\_torpedoes
      7. getCabbage(): return m\_CabbagePoints
      8. addTorpedoes(): add 5 to m\_torpedoes
      9. goodiePickedUp(): increase score by 100, play sound, and decide what to do based on goodie picked up
   6. Projectile
      1. Projectile(): abstract constructor
      2. ~Projectile(): virtual destructor
      3. doSomething(): call move() unless offscreen, if so, set dead
      4. isOffScreen(): if not within the field of play, return true else return false
      5. move(): pure virtual because all projectiles move but not the same way
   7. Cabbage
      1. Cabbage(): constructor
      2. ~Cabbage(): virtual destructor
      3. Move(): move 8 pixel to the right and rotate 20 degrees
   8. Torpedo
      1. Torpedo(): constructor
      2. ~Torpedo(): virtual destructor
      3. Move(): move 8 pixel to the right if shot from player and 8 pixel to the left if shot from Torpedo
   9. Turnip
      1. Turnip(): constructor
      2. ~Turnip(): virtual destructor
      3. Move(): move 6 pixel to the left and rotate 20 degrees
   10. Alien
       1. Alien(): set m\_travel\_speed, m\_flight\_plan\_length, and m\_direction
       2. ~Alien(): virtual destructor
       3. doSomething(): prompts Alien to do something, check for collision, set new flight plan, do random behavior, and move
       4. move(): determines what direction to move
       5. sufferDamage(): determines how much damage to suffer and if health is 0 or lower, call setDead ()
       6. getTravelSpeed(): return m\_travel\_speed
       7. getFlightPLanLength(): return m\_flight\_plan\_length
       8. getDirection(): return m\_direction
       9. needNewFlightPlan(): return whether or not a new plan is needed
       10. setFlightPLanLength(): set m\_flight\_plan\_length to specified length
       11. setTravelSpeed(): set m\_travel\_speed to specified speed
       12. setDirection(): set m\_direction to direction
       13. withinRangeNB(): return whether or not player is within 4 pixels up and down and to the left of the Alien
       14. deathSequence() and withinRangeAction() are pure virtual because all Aliens have do something when they died and do something if within range, but all do something differently
   11. Smallgon
       1. Smallgon(): constructor
       2. ~Smallgon(): virtual destructor
       3. withinRangeAction(): returns whether or not Smallgon shot a turnip
       4. deathSequence(): increase score, play Sound, and add explosion
   12. Smoregon
       1. Smoregon(): constructor
       2. ~Smallgon(): virtual destructor
       3. withinRangeAction(): returns true if Smoregon shot turnip, else chance to change flight plan and/or return false
       4. deathSequence(): chance to drop a repair or torpedo goodie
   13. Snagglegon
       1. Snagglegon(): set Direction to 2 and travelSpeed to 1.75
       2. ~Snagglegon(): virtual destructor
       3. withinRangeAction(): returns whether or not Snagglegon shot a torpedo
       4. deathSequence(): chance to drop a life goodie
   14. Goodie
       1. Goodie(): constructor
       2. ~Goodie(): virtual destructor
       3. doSomething(): move unless off screen, which case set dead
       4. move(): move down and left 0.75 pixels in both directions
   15. All Goodies are the same, just with different imageID’s
   16. Explosion
       1. Explosion(): set m\_frameNumber to 0
       2. ~Explosion(): virtual destructor
       3. doSomething(): increase size by 1.5 and call move
       4. move(): increase m\_frameNumber, if it is greater than or equal to 4, call setDead()
2. I have no noteworthy bugs, at least that I am aware of
3. Design decisions
   1. The specs kept talking about the level up bonus, but the distributed game didn’t have a level up bonus and the exact points added was never stated. Therefore, I decided not to add a level up bonus.
   2. I used imageID’s to detect collisions and such because I thought it was the easiest way to differentiate objects because type casting seemed like a pain.
   3. I used the NachenBlaster object to handle collisions between enemy missiles and goodies. I used Alien objects to handle collisions between the NachenBlaster and player fired missiles. I did this because it would allow me to handle collision damage most effectively. Also, I thought one-way collision detection would be most simple and beneficial.
   4. Continuing with that, I decided to let the NachenBlaster handle picking up Goodies because the Goodies basically all behave the same outside of looking different. Truth be told, I actually only need one goodie class and just differentiate them using images.
4. Testing
   1. Actor, Projectile, Ship, and Alien are abstract classes, so they were tested using inheritance
   2. Star: Tested using the init function by adding 30 stars into a m\_actors vector. Using the move function, I tested that each star would move to the left. The move function also tested whether or not a new star would be added on a 1/15 probability. Finally, the move function also tests whether or not a star gets deleted if it reaches the far left side of the screen.
   3. NachenBlaster: Tested keyboard movement by running the game and moving the player in different direction. All keyboard functions run appropriately and player is also able to shoot cabbages and torpedoes. Collision was tested using a single Alien and ramming the Alien caused the Player to destroy the Alien and take health damage. I also had aliens shoot turnips and torpedoes at the NachenBlaster. Finally, the NachenBlaster was able to pick up goodies when I added goodies into the game.
   4. Aliens (Smoregon, Smallgon, Snagglegon): Tested all Alien subclasses by putting them into the game and observing their normal behavior. All Aliens behaved normally without the player, moving in the appropriate directions and changing directions when need be. Once the player was added, each Alien’s random behavior was also handled correctly. Collision was tested with the NachenBlaster and its projectiles. Aliens did not collide with anything it wasn’t supposed to collide with. Aliens also received the correct amount of damage based on what it collided with. I placed each type of alien individually in the game world and tested each of its individual behaviors.
   5. Projectiles (Cabbage, Turnip, Torpedo): Tested all Projectiles subclasses by putting them into the world when called upon by their shooters. Cabbages where correctly shot from the player and turnips from the Aliens. Snagglegons shot Torpedoes from the right direction and so did the player. The Projectile moved at the correct speeds when placed by themselves in the world and were removed from the world when they traveled off screen.
   6. Goodies (Repair, Life, Health): Tested all Goodie subclasses by putting them into the world by themselves and when called upon when Aliens were destroyed. They all moved in the appropriate direction and at the appropriate speed. There really isn’t anything else to test in my implementation of the game.
   7. Explosion: Tested Explosion by adding it individually In the game. It correctly grew by the right size scaling and was removed after the fourth tick. Then, I collided the player into some Aliens and the Explosion was correctly added into the world by the Aliens.