CMSC 3103 Object Oriented Software Design & Construction

Fall 2015

Daniel Ware
***20359025**
Applied Mathematics and Computer Science

Section I: User Stories

As a player, I want to press start so that the game begins.
As a player, I want to press Quit so that the game exits.
As a player, I want to move the mouse to rotate the “shooter” to face the mouse.
As a player, I want to press the left arrow key so that the “shooter” moves to the left 5 pixels.
As a player, I want to press the ‘a’ key so that the “shooter” moves to the left 5 pixels.
As a player, I want to press the right arrow key so that the “shooter” moves to the right 5 pixels.
As a player, I want to press the ‘d’ key so that the “shooter” moves to the right 5 pixels.
As a player, I want to press the up arrow key so that the “shooter” moves up 5 pixels.
As a player, I want to press the ‘w’ key so that the “shooter” moves up 5 pixels.
As a player, I want to press the down arrow key so that the “shooter” moves down 5 pixels.
As a player, I want to press the ‘s’ key so that the “shooter” moves down 5 pixels.
As a player, I want to press the left mouse button so that the “shooter” fires a missile
As a player, I want to move the “shooter” to intersect a power up so that I can deplete it.
As a player, I want to shoot missiles so that they can destroy UFOs.
As a player, I want to shoot missiles so that they can destroy falling bombs.
As a more powerful player, I want to shoot lasers so that they can destroy UFOs.
As a more powerful player, I want to shoot lasers so that they can destroy falling bombs.
As a player, I want to destroy UFOs with missiles so that I can get a higher score.
As a player, I want to destroy falling bombs with missiles so that they do not destroy me.
As a player, I want to destroy UFOs with myself so that I can get a higher score.
As a player, I want to destroy falling bombs with myself so that they do not destroy me.
As a player, I want to fire a missile that will travel to my mouse position and explode.
As a player, I want to see UFOs spawn in the upper half of the screen.
As a player, I want to see UFOs bounce off the left and right side of the screen.
As a player, I want to see bombs fall from the top in a zig-zag pattern.
As a player, I want to see bombs explode when they are shot.
As a player, I want to see UFOs “fall” when they are shot.
As a player, I want to see power ups deplete after I have used them.
As a player, I want to see power ups deplete after a UFO or bomb has used them.
As a player, I want to UFOs become larger after depleting a power up.
As a player, I want to bombs become larger after depleting a power up.
As a player, I want to deplete power ups to get a higher score.
As a player, I want to deplete 10 power ups to get a laser upgrade and become a more powerful player.
As a player, I want to see the background scroll.
As a player, I want to move the mouse and have the “shooter” nose follow the cursor.

Section II: CRC Cards

Animator	
Responsibilities	Collaborators
Processes collision Runs game loop Updates Score	Quadtree GameData, GamePanel MainWindow

ButtonListener	
Responsibilities	Collaborators
Starts Game Exits Game	GameData Animator

KeyController	
Responsibilities	Collaborators
Moves Shooter	Shooter

Main	
Responsibilities	Collaborators
Calls for GamePanel creation Starts Animator loop	GamePanel Animator

MouseController	
Responsibilities	Collaborators
Aim and fire missiles and lasers Rotate Shooter	Missile, Laser, Weapon Component Shooter

MouseMovementListener	
Responsibilities	Collaborators
Contains mouse movement data for rotating the shooter	Shooter

Quadtree	
Responsibilities	Collaborators
Handles all game objects for collision detection Inserts game objects into the tree Returns game objects in the same quad Can be cleared	GameObject

Bomb	
Responsibilities	Collaborators
Rendered as rounded bomb image on screen Keep track of number of bombs alive Become larger when intersecting a PowerUp Has a collision box Keep track of BombState Track logic for going to next state	PowerUp CollisionBox BombState

FlyingSaucer	
Responsibilities	Collaborators
Render as an UFO image on screen Keep track of number of UFOs alive Become larger when intersecting a PowerUP Has a collision box Keep track of SaucerState Track logic for going to next state	PowerUp CollisionBox SaucerState

GameData	
Responsibilities	Collaborators
Initialize game objects on screen Keep game objects at a constant number Initialize moving background Initialize weapon	Shooter, Bomb, PowerUp Background WeaponComponent, BasicWeapon

GameFigure	
Responsibilities	Collaborators
Abstract class for all game objects	

Missile	
Responsibilities	Collaborators
Render as a small circle on screen Keep track of location Keep track of size Has a collision box Keep track of MissileState Track logic for going to next state	CollisionBox MissileState

PowerUp	
Responsibilities	Collaborators
Render as a small rounded crate on screen	CollisionBox
Keep track of number of power ups alive	PowerState
Has a collision box	
Keep track of PowerState	
Track logic for going to next state	

Shooter	
Responsibilities	Collaborators
Render as space ship image on screen	Missile
Keep track of number of power ups collected	CollisionBox
Keep track of location to fire weapon	GameData
Keep track of ship rotation	
Keep track of weapon power	
Has a collision box	

GamePanel	
Responsibilities	Collaborators
Renders the game objects in the game window	MainWindow, GameData

MainWindow	
Responsibilities	Collaborators
Initializes main game window	MouseController, ButtonListener, KeyController

CollisionBox	
Responsibilities	Collaborators
Interface for a bounding collision box	

Laser	
Responsibilities	Collaborators
Render as a red line from shooter nose to clicked position	CollisionBox
Has a collision box	LaserState
Keep track of decay time	
Keep track of LaserState	
Track logic for going to next state	

LaserState	
Responsibilities	Collaborators
Interface for all laser states	

LaserStateAlive	
Responsibilities	Collaborators
State that represents that a laser has been fired Set state to LaserStateDecay	LaserState, LaserStateDecay

LaserStateDecay	
Responsibilities	Collaborators
State that represents that a laser is decaying Set state to LaserStateDone	LaserState, LaserStateDecay

LaserStateDone	
Responsibilities	Collaborators
State that represents that a laser is ready to be removed from the game	LaserState

BombState	
Responsibilities	Collaborators
Interface for all bomb states	

BombStateAlive	
Responsibilities	Collaborators
State that represents that a bomb has been created and is falling in a zig-zag Set state to BombStateExploding	BombState, BombStateExploding

BombStateExploding	
Responsibilities	Collaborators
State that represents that a bomb is exploding Set state to BombStateDone	BombState, BombStateDone

BombStateDone	
Responsibilities	Collaborators
State that represents that a bomb is ready to be removed from the game	BombState

MissileState	
Responsibilities	Collaborators
Interface for all missile states	

MissileStateAlive	
Responsibilities	Collaborators
State that represents that a missile has been fired Set state to MissileStateExploding	MissileState, MissileStateExploding

MissileStateExploding	
Responsibilities	Collaborators
State that represents that a missile has reached its destination or has collided with an enemy and is exploding Set state to MissileStateDone	MissileState, MissileStateDone

MissileStateDone	
Responsibilities	Collaborators
State that represents that a missile is ready to be removed from the game	MissileState

SaucerState	
Responsibilities	Collaborators
Interface for all UFO states	

SaucerStateAlive	
Responsibilities	Collaborators
State that represents that a UFO has been created and is moving in the game Set state to SaucerStateDying	SaucerState, SaucerStateDying

SaucerStateDying	
Responsibilities	Collaborators
State that represents that a UFO has been hit by a missile or laser and is “falling” Set state to SaucerStateDone	SaucerState, SaucerStateDone

SaucerStateDone	
Responsibilities	Collaborators
State that represents that a UFO is ready to be removed from the game	SaucerState

PowerState	
Responsibilities	Collaborators
Interface for all power ups	

PowerStateAlive	
Responsibilities	Collaborators
State that represents that a power up has been created and is moving in the game Set state to PowerStateUsed	PowerState, PowerStateUsed

PowerStateUsed	
Responsibilities	Collaborators
State that represents that a power up has collided with an enemy or shooter and has been used up Set state to PowerStateDone when this object has moved off the screen	PowerState, PowerStateDone

PowerStateDone	
Responsibilities	Collaborators
State that represents that a power up is ready to be removed from the game	PowerState

WeaponComponent	
Responsibilities	Collaborators
Interface for all shooter weapons	

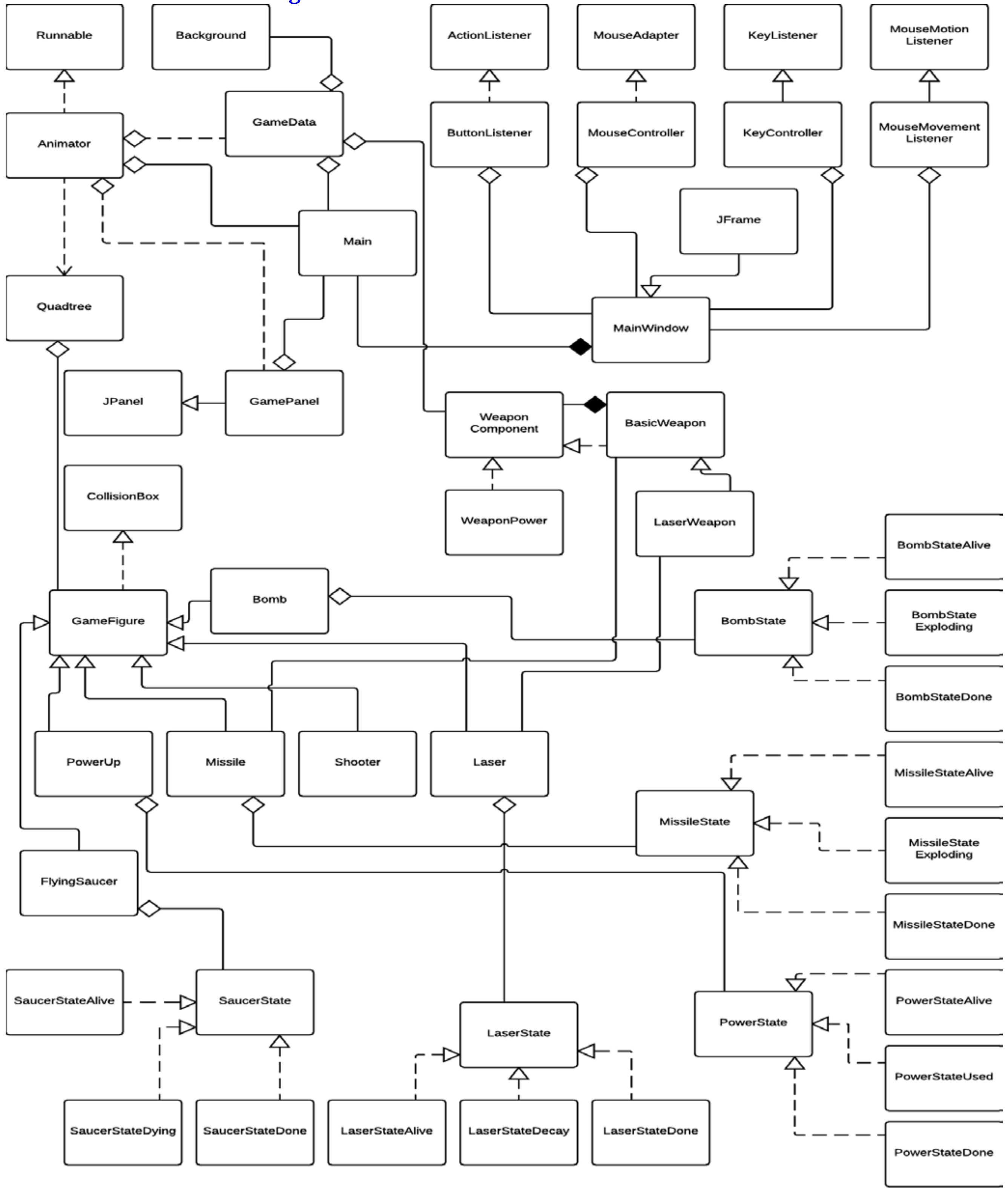
WeaponPower	
Responsibilities	Collaborators
Abstract class that Parents all weapon types Is called to polymorph the ability for the shooter to fire	WeaponComponent

BasicWeapon	
Responsibilities	Collaborators
Creates a new Missile at the shooters location	WeaponComponent, Missile, WeaponPower

LaserWeapon	
Responsibilities	Collaborators
Creates a new Laser at the shooters location	Missile, Laser, WeaponPower

Background	
Responsibilities	Collaborators
Render the background images Keep track of image movement	GamePanel

Section III: UML Class Diagram



Section IV: Implementation of Required Features

(1) Implementation of One Additional Enemy Type

The additional enemy type that I implemented was a falling bomb. It has a “bomb” image. It moves from above the screen, to the bottom of the screen. It also moves in a zig-zag pattern left and right to make it more difficult to hit with a laser or missile. When it is hit by a missile or laser or the “shooter” then it plays an explosion animation. The explosion animation consists of 2 parts. There is an explosion and then an implosion, both with different images to represent them. It also becomes smaller as the animation plays. The bomb enemy can use power-ups when it collides with them to become larger. The images scale up for this.

(2) Destroying Effects of Enemies

The bomb enemy type has an animation that looks like it is exploding. This consists of 2 images that are played at certain times make it look like an explosion.

The UFO enemy type has an animation that makes it look like it is falling from the sky. It is done by controlling the size and position of the UFO. When it is hit, it moves in the positive y axis and the image is gradually scaled to a small image.

The power up enemy type is rendered as a small crate with yellow stripes on it. When it is used by either an enemy or shooter, then it is represented as a small white ball and it continues to move off the screen out into space.

(3) Implementation of the State design pattern for three Enemy types

The state design pattern was used to control the states of each enemy. Each object had it’s own context to keep track of its state and referred to it when needing to know how to behave. As you can see in Section VI, each State needed to implement a method to go to the next state when the logic told it to.

(4) Implementation of an additional design pattern

The additional design pattern I chose to implement was the decorator design pattern. I used this to attach additional power to the shooter’s ability to fire. When a certain number of power ups were used, the laser ability was added to the shooter weapon. Then both missiles and laser would fire when clicked.

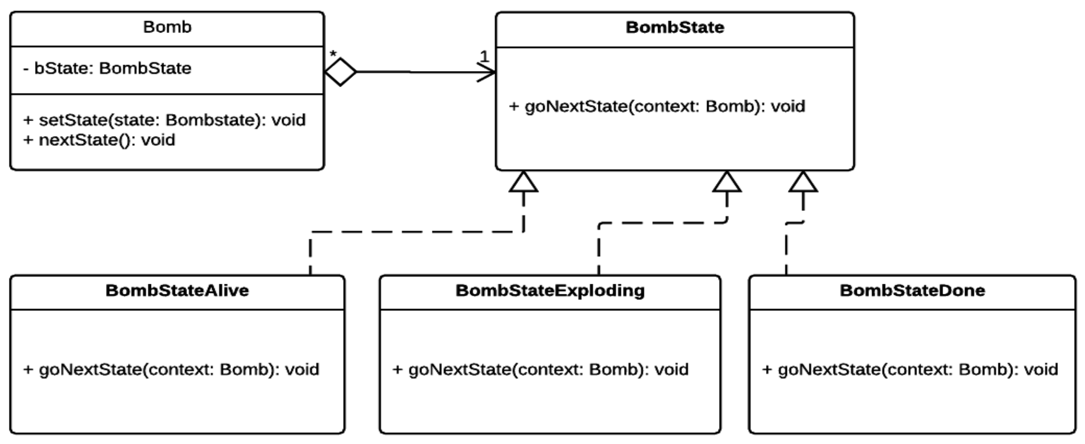
(5) Implementation of an optional feature

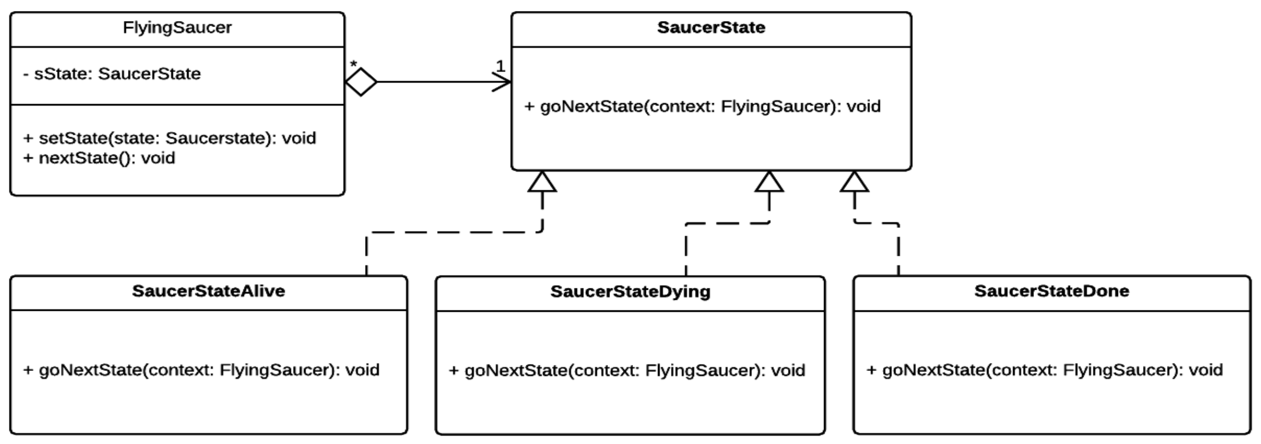
The optional feature I implemented was a quadtree. The quadtree seperates the screen into “quads” (rectangles) in a tree structure. Each node of the tree can have 4 children. This significantly reduces cpu usage for collision detection because only things in the same quad need to be checked for collision. Objects on the opposite side of the screen could not collide with objects far away so it is inefficient to check for that. The concrete implementation of the quadtree involves adding all objects to the quadtree. The screen is then broken into rectangles depending on the number of objects in a quad. Then for each gameobject I send to the quad tree, it sends back an array of game objects in the same quad. Then I can check collision for those object only and move on. The quadtree has to be emptied and refilled each turn of the game logic.

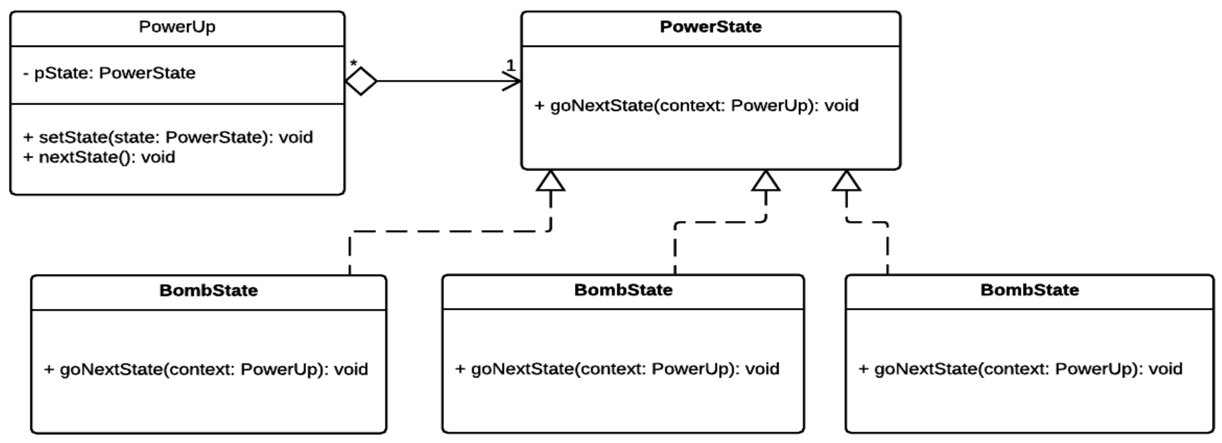
(6) New Ideas of the Game

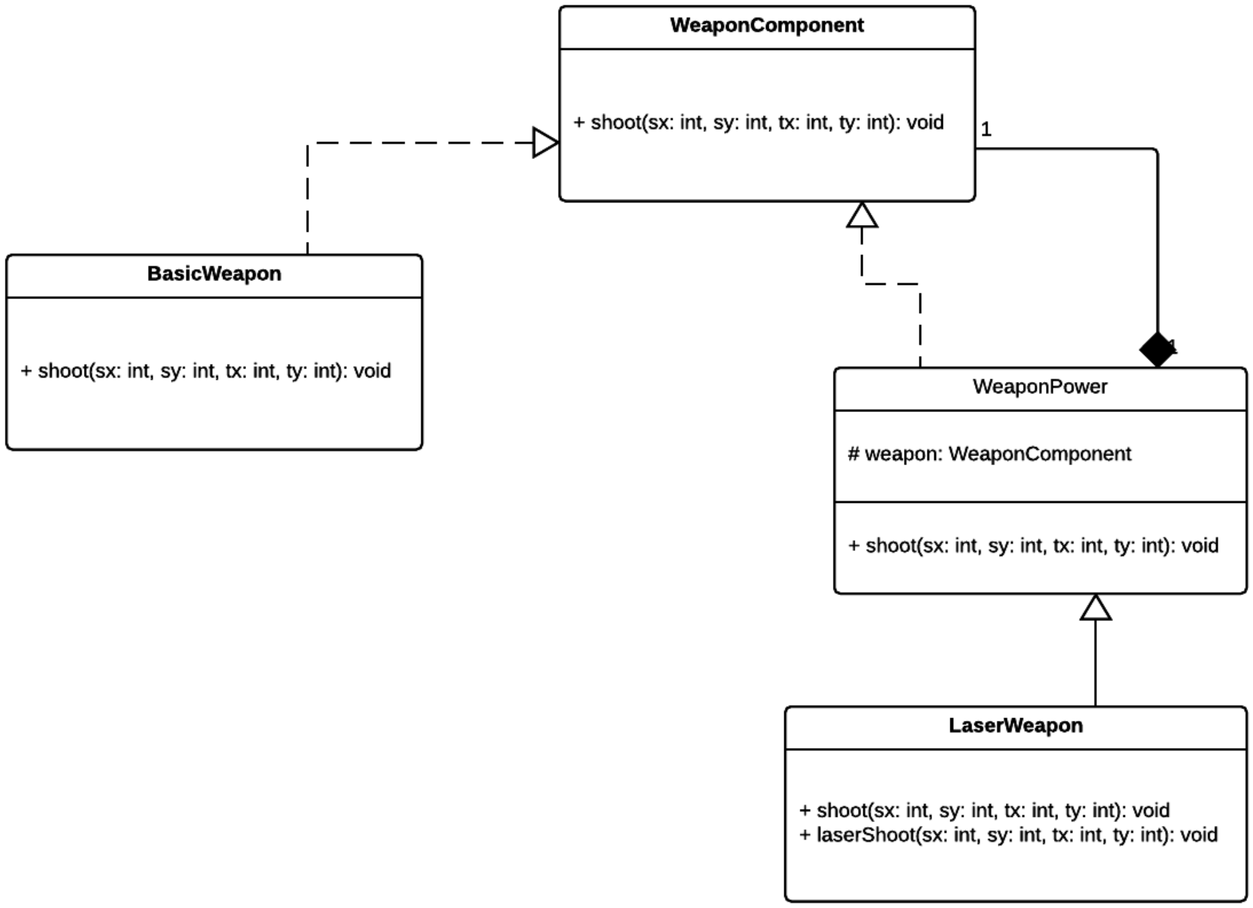
The main shooter rotates to follow the mouse cursor and the background scrolls. The use of the original “bombs” have been repurposed to a different “enemy type” the power up. A new enemy bomb was created and has completely different behavior. The collecting and using power ups allow for a upgrade to a laser weapon.

Section V: UML Diagrams for Design Patterns Implemented


Bomb
- bState: BombState
+ setState(state: Bombstate): void
+ nextState(): void
*
1
BombState
+ goNextState(context: Bomb): void
BombStateAlive
+ goNextState(context: Bomb): void
BombStateExploding
+ goNextState(context: Bomb): void
BombStateDone
+ goNextState(context: Bomb): void


FlyingSaucer
- sState: SaucerState
+ setState(state: Saucerstate): void
+ nextState(): void
*
1
SaucerState
+ goNextState(context: FlyingSaucer): void
SaucerStateAlive
+ goNextState(context: FlyingSaucer): void
SaucerStateDying
+ goNextState(context: FlyingSaucer): void
SaucerStateDone
+ goNextState(context: FlyingSaucer): void


PowerUp
- pState: PowerState
+ setState(state: PowerState): void
+ nextState(): void
*
1
PowerState
+ goNextState(context: PowerUp): void
BombState
+ goNextState(context: PowerUp): void
BombState
+ goNextState(context: PowerUp): void
BombState
+ goNextState(context: PowerUp): void


WeaponComponent
+ shoot(sx: int, sy: int, tx: int, ty: int): void
1
BasicWeapon
+ shoot(sx: int, sy: int, tx: int, ty: int): void
1
WeaponPower
# weapon: WeaponComponent
+ shoot(sx: int, sy: int, tx: int, ty: int): void
LaserWeapon
+ shoot(sx: int, sy: int, tx: int, ty: int): void
+ laserShoot(sx: int, sy: int, tx: int, ty: int): void

Section VI: Proof for the Correctness of Implemented Design Patterns

State	**The original specification**	**Your implementation**
Intent	Allow an object to alter it's behavior when it's internal state changes. The object will appear to change its class.	In the game, the bomb type has its behavior determined by the 'state' it is in.
Context	Defines the interface of interest to clients Maintains an instance of a concrete state subclass that defines the current state	Bomb – This class has a bState variable that holds the current state of the bomb. It also has the logic for what to do in each state.
State	Defines an interface for encapsulating the behavior associated with a particular state of the Context	BombState – Void goNextState(Bomb context) method is the interface we will use to go to the next state
Concrete State	Each subclass implements a behavior associated with a state of the context	BombStateAlive, BombStateExploding, BombStateDone goNextState is overridden in each state to set the state to the next state BombStateDone sets a flag so the game logic knows that it is ready to be removed from game data

Decorator	**The original specification**	**Your implementation**
Intent	Attach additional responsibilities to an object dynamically	In the game, the shooter's fire ability has a laser power added to it when a certain number of power ups are used.
Component	Defines the interface for objects that can have responsibilities added to them dynamically	WeaponComponent – Void shoot(int, int, int, int) method is the interface we will use to access the responsibilities of the concrete component
Concrete Component	Defines an object to which additional responsibilities can be attached	WeaponPower - Void shoot(int, int, int, int) method is overridden to call the weapon to shoot
Decorator	Maintains a reference to a component object and defines an interface that conforms to component's interface	BasicWeapon – void shoot(int, int, int, int) is overridden to fire a missile when shoot is called from the game logic
Concrete Decorator	Adds responsibilities to the component	LaserWeapon- Void shoot(int, int, int, int) method is overridden to call super.shoot(int, int, int. int) and laserShoot(int, int, int.int)