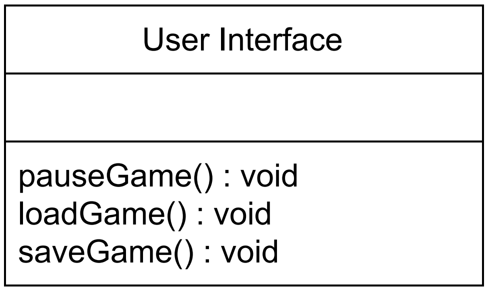
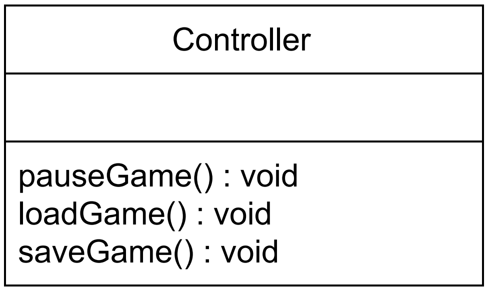
Detailed Pattern Document

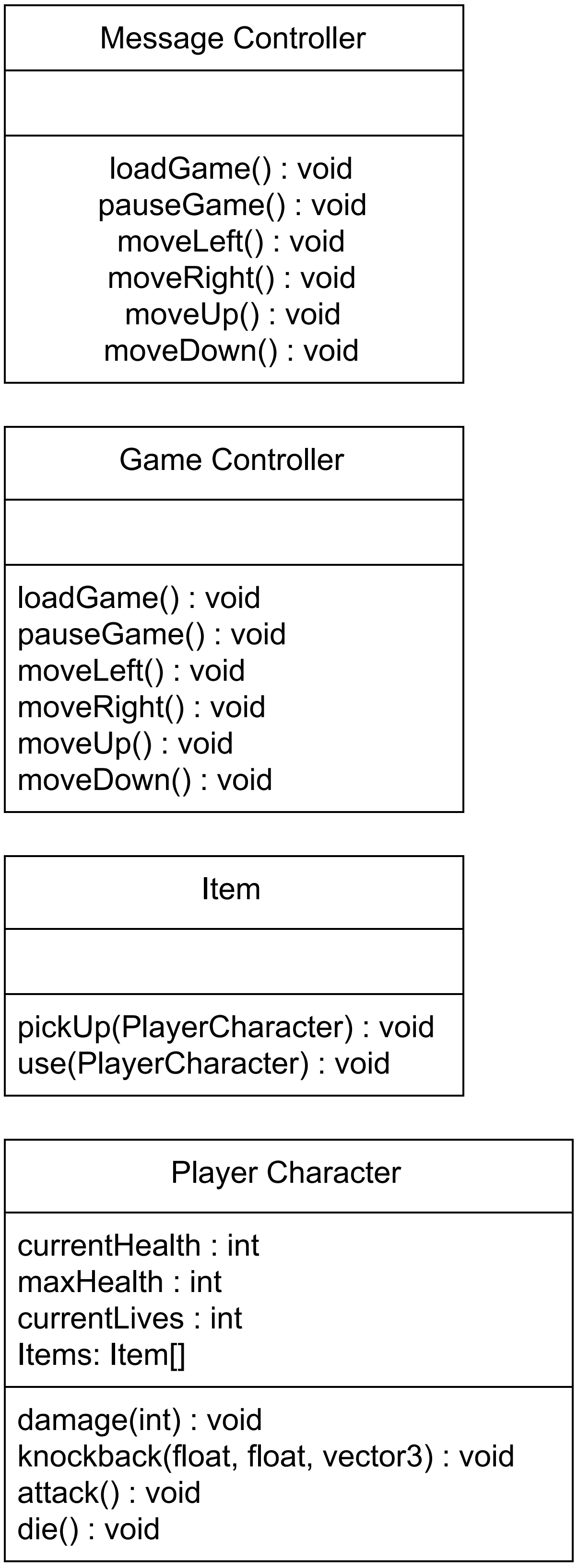
Pattern: Structured Composite

Why:

For Burger Breakout we thought it would be best to go with a Structured Composite design. This is because we already know all the objects and relations that will be in the game, so it would be more easily understood in a structured setting. The composite choice was because the game is built off classes of objects that diverge into smaller more specific objects. For example we have an “Item” class that holds all the items objects in the game. Each object has particular functions and operations that are specific to itself. Using this layout we can more easily visualize the different interactions each class of objects will have with each other. We have not entirely implemented this pattern into our code yet, but it is something we want to work towards in the future.



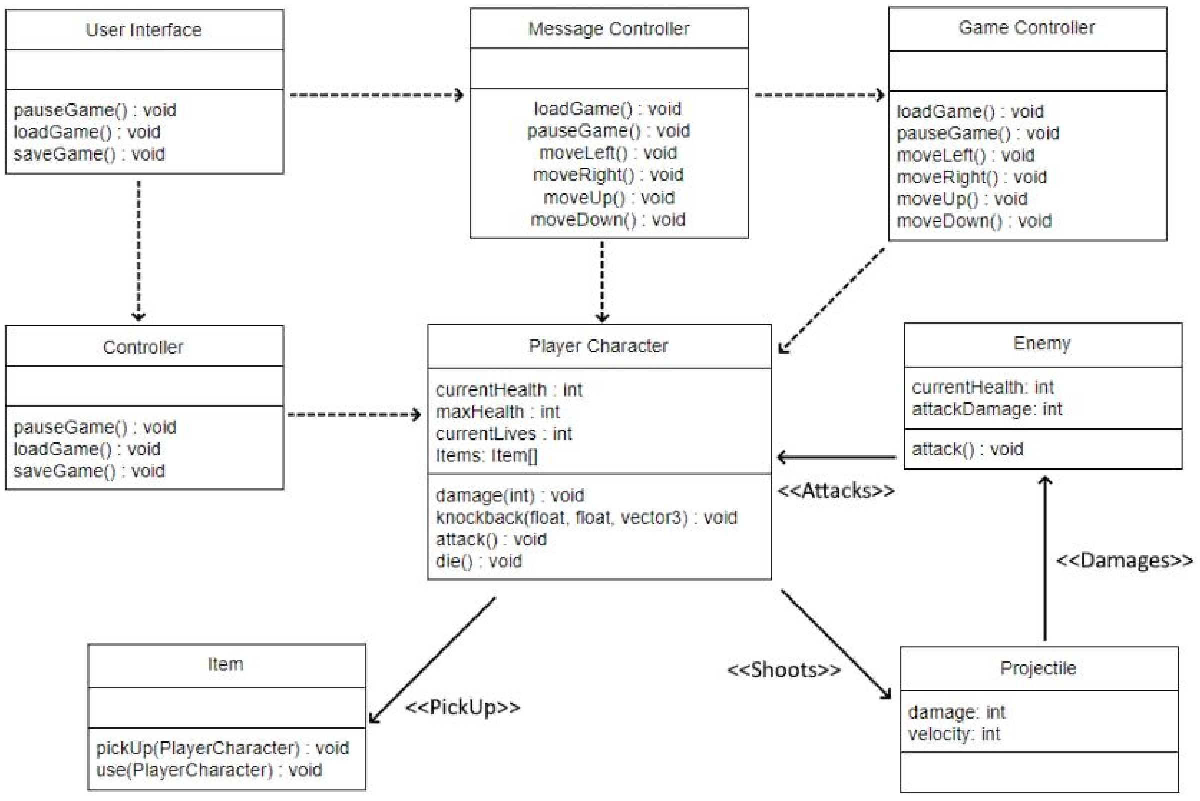



Message Controller
loadGame() : void
pauseGame() : void
moveLeft() : void
moveRight() : void
moveUp() : void
moveDown() : void
Game Controller
loadGame() : void
pauseGame() : void
moveLeft() : void
moveRight() : void
moveUp() : void
moveDown() : void
Item
pickUp(PlayerCharacter) : void
use(PlayerCharacter) : void
Player Character
currentHealth : int
maxHealth : int
currentLives : int
Items: Item[]
damage(int) : void
knockback(float, float, vector3) : void
attack() : void
die() : void

Projectile
damage: int velocity: int

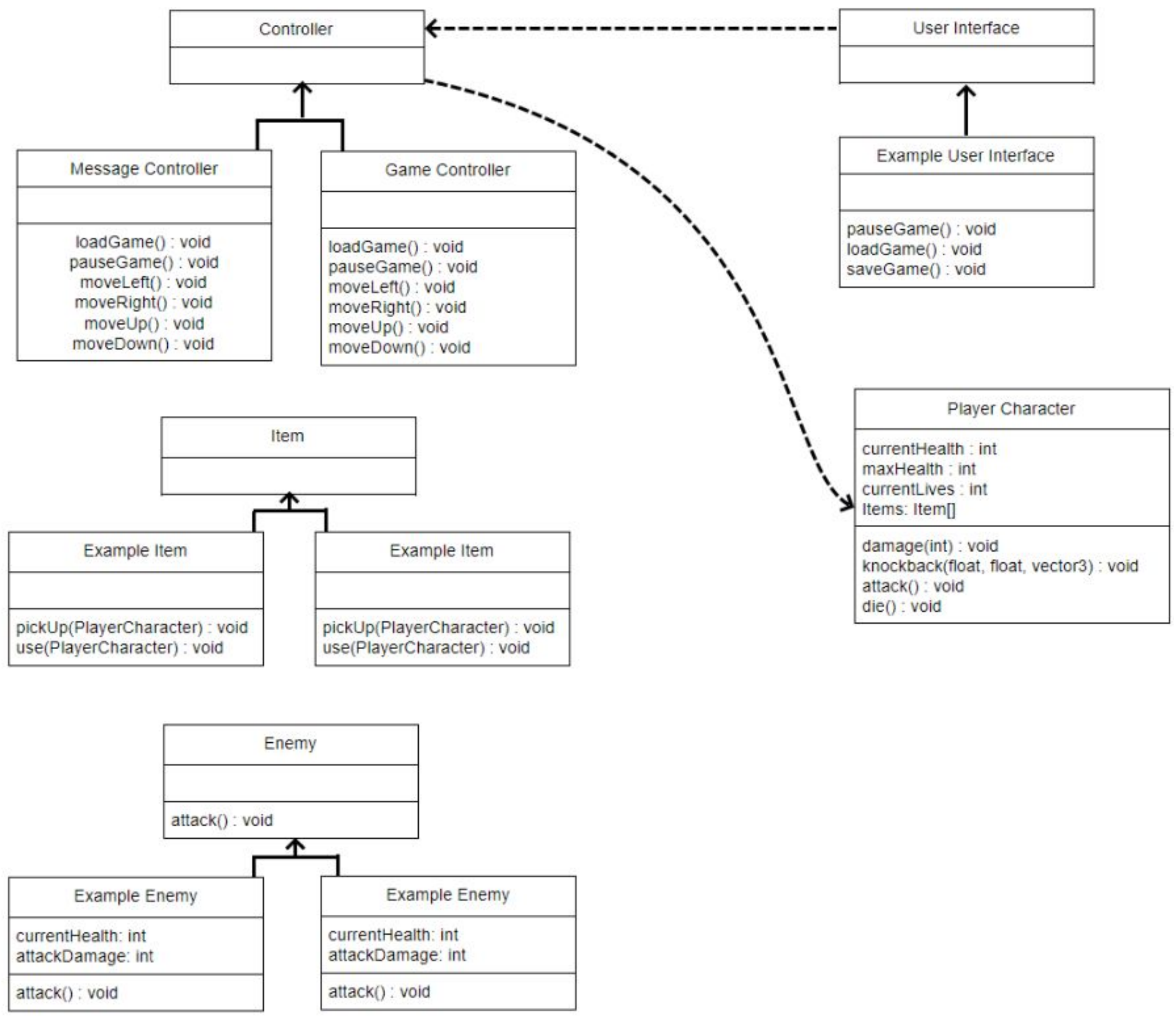
Enemy
currentHealth: int attackDamage: int
attack() : void

Detailed Design:



Ideal Structured Composite Design:

This is **NOT** what we currently have implemented, but an overview of what we want to strive for in our game's design pattern. In this design, objects are parts of larger classifications making the entire system much easier to understand.


Controller
User Interface
Message Controller
loadGame() : void
pauseGame() : void
moveLeft() : void
moveRight() : void
moveUp() : void
moveDown() : void
Game Controller
loadGame() : void
pauseGame() : void
moveLeft() : void
moveRight() : void
moveUp() : void
moveDown() : void
Example User Interface
pauseGame() : void
loadGame() : void
saveGame() : void
Player Character
currentHealth : int
maxHealth : int
currentLives : int
Items: Item[]
damage(int) : void
knockback(float, float, vector3) : void
attack() : void
die() : void
Item
Example Item
pickUp(PlayerCharacter) : void
use(PlayerCharacter) : void
Example Item
pickUp(PlayerCharacter) : void
use(PlayerCharacter) : void
Enemy
attack() : void
Example Enemy
currentHealth: int
attackDamage: int
attack() : void
Example Enemy
currentHealth: int
attackDamage: int
attack() : void