Shop Design Group 66

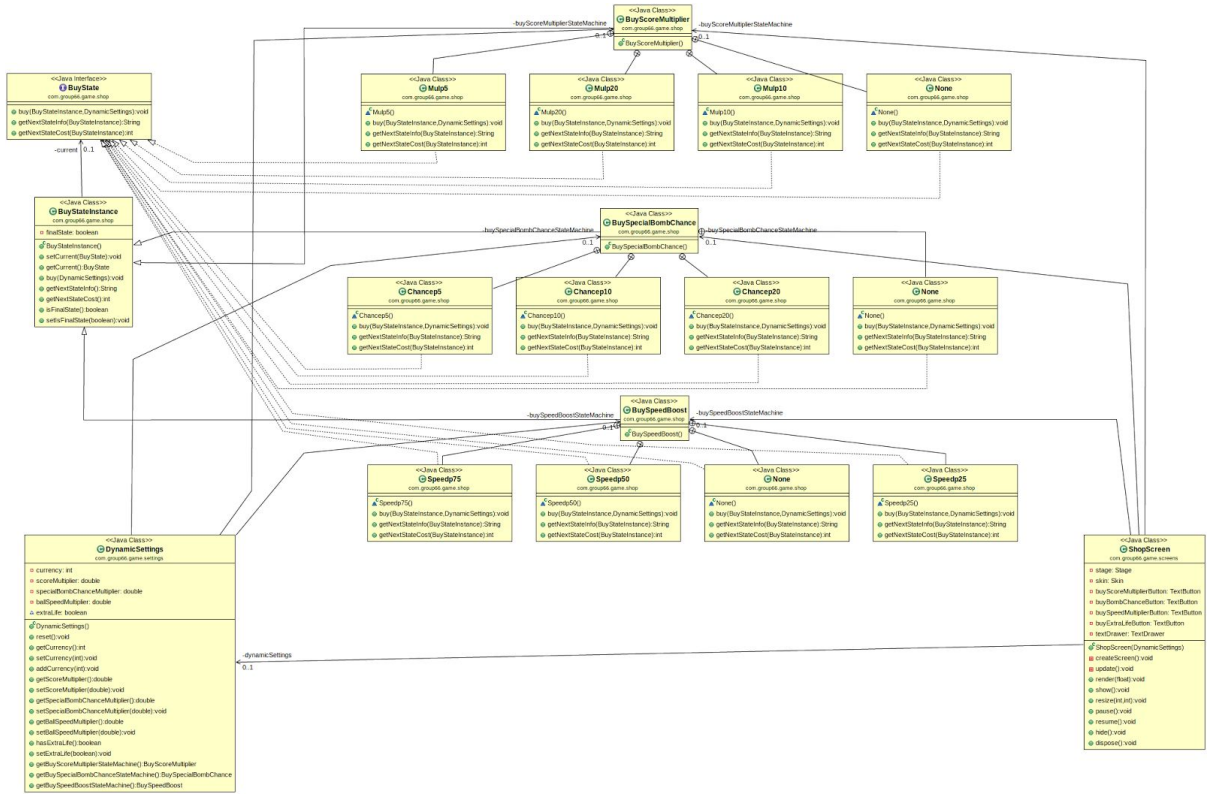
This document describes the design of a shop and related components for use within the Bust-A-Move game using responsibility driven design and UML.

Class Responsibility Collaborator (CRC)

Class	**Responsibility**	**Collaborators**
DynamicSettings	Manage settings that can change at runtime	
BuyState	Interface for the states	
BuyStateInstance	The core of the state machines	Uses the BuyState interface for the state definition.
BuyScoreMultiplier	State machine for the Score Multiplier option. Manages the current state.	Inherits from BuyStateInstance. Uses settings from DynamicSettings
BuySpecialBombChance	State machine for the Special Bomb Chance option. Manages the current state.	Inherits from BuyStateInstance. Uses settings from DynamicSettings
BuySpeedBoost	State machine for the Speed Boost option. Manages the current state.	Inherits from BuyStateInstance. Uses settings from DynamicSettings
ShopScreen	Manages the shop game screen	Uses the BuyScoreMultiplier, BuySpecialBombChance and BuySpeedBoost state machines. Uses and sets settings in DynamicSettings

UML diagram

The UML diagram of the created classes for this assignment. A png image of this class diagram is also added separately.



Technical Design and Implementation

The created shop uses the state design pattern to keep track of location of the player in the design chain. The upgrades/power-ups of the same kind must be bought sequentially. This is implemented for the special ball chance increase, the ball speed increase and the score multiplier power-up. The state machines inherit from the BuyStateInstance class, which is a start class with the basic functionality of the state machine. This is used to prevent unneeded code duplication for the 3 state machines. The Extra Life power-up is also available, but this is a boolean value and implementing this as a state pattern does not provide any benefits. The DynamicSettings class was introduced to keep track of settings which are used throughout the game; which can change at runtime as opposed to static settings which are defined in the settings.Config class.

In the shop screen, the state machines are combined with the buttons, this makes it that when a button is pressed the state goes to the next state if enough currency is available.

For the splitscreen multiplayer the dynamic settings are shared. The currency update takes the average of the earned currency of all the players and the bought power-ups are shared.