A7 SDD-3

Team 10 – Zombie Apocalypse

Chad Nelson, Cole Anagnost, Tas Fox, Tim Flannigan

Team Work Distribution Form

Assignment A7
Group #10

Name	% of Effort	Extra Work	Description of Work
Chad Nelson	25%		SDD-3 Assignment
Cole Anagnost	25%		SDD-3 Assignment
Tas Fox	25%		SDD-3 Assignment
Tim Flanigan	25%		SDD-3 Assignment

Meeting Minutes

Name	Present	Late > 5 min	Informed of Absence?	Scribe?
Chad Nelson	Yes	No	-	-
Cole Anagnost	Yes	No	-	Yes
Tas Fox	Yes	No	-	-
Tim Flanigan	Yes	No	-	-

Com

Name	Old Action Item	Status	New Action Item
Chad Nelson	SDD-3 Assignment	Complete	
Cole Anagnost	SDD-3 Assignment	Complete	
Tas Fox	SDD-3 Assignment	Complete	
Tim Flanigan	SDD-3 Assignment	Complete	

Software Design Description

Zombie Apocalypse

Chad Nelson
Cole Anagnost
Tasewell Fox
Tim Flanigan

TABLE OF CONTENTS

REVISION HISTORY

Version	Date	Author	Change
0.1	10/29/2009	Group 10	Initial Document
0.2	11/2/2009	Group 10	SDD-1 Sections Completed
0.3	11/6/2009	Group 10	SDD-2 Sections Completed

1. INTRODUCTION

1.1 PURPOSE

The purpose of this software design description is to describe the architecture and design of Zombie Apocalypse. The intended audience for this document is its developers, Professor Simanta Mitra, TAs, and the students of ComS 309 Fall 2009.

1.2 SCOPE

The scope of this document is limited to the low level architectural and design considerations for Zombie Apocalypse. This document should provide its reader with a detailed understanding of the game, expected user interaction, and how the system works.

1.3 DEFINITIONS, ACRONYMS, ABBREVIATIONS

Term	Description
Alpha	Transparency
AS3	(ActionScript 3)The programming language used with Adobe Flash
Chat Room	A collection of client connections to the server that are grouped together for the purpose of sharing text messages; the lobby and all game rooms are chat rooms
Client	An instance of the game run via a Flash virtual machine; there are many clients
DisplayObject	An interface that any visible item implements
Display Tree	A tree containing a root node and it's children; any visible object must be attached to the display tree before it is visible on the stage
Game Room	A chat room where all players have the intent of playing a multiplayer game together; limited number of players; there can be any number of game rooms
Lobby	The default chat room where players are allowed to create/join game rooms; unlimited number of players
MovieClip	An AS3 object that implements the DisplayObject interface; contains a number of frames that are animated
Player	(Actor) A person that plays the game by running an instance of the client
Server	A program that manages connections and stores information. Manages room creation and multiplayer communication, and stores data about player accounts, high scores, and game status.
Sprite	An AS3 object that implements the DisplayObject interface; contains a single static image
Stage	A static AS3 object that represents the view of all visible objects

*.SWC	(Shockwave Flash Component extension) Contains code and graphics that can be instantiated by another Flash program
*.SWF	(Shockwave Flash extension) A Flash movie file; Contains compiled byte code and graphics that will be interpreted by a Flash Player and displayed on a webpage

1.4 DESIGN GOALS

1. Usability - The game must provide an intuitive player interface and control scheme, in addition to help or tips for an inexperienced players. The game should also remain responsive during intensive processing for many onscreen opponents. This design goal is met by implementing a widely used control scheme (WASD keyboard controls for player movement with QER for other actions, mouse for aiming and firing) with all necessary information shown in a GUI overlay.
2. Multiplayer - The game must allow multiple players to play on the same map in real time. This design goal will be met using a client-server architecture, where central servers store the player's data and relay information between client machines.

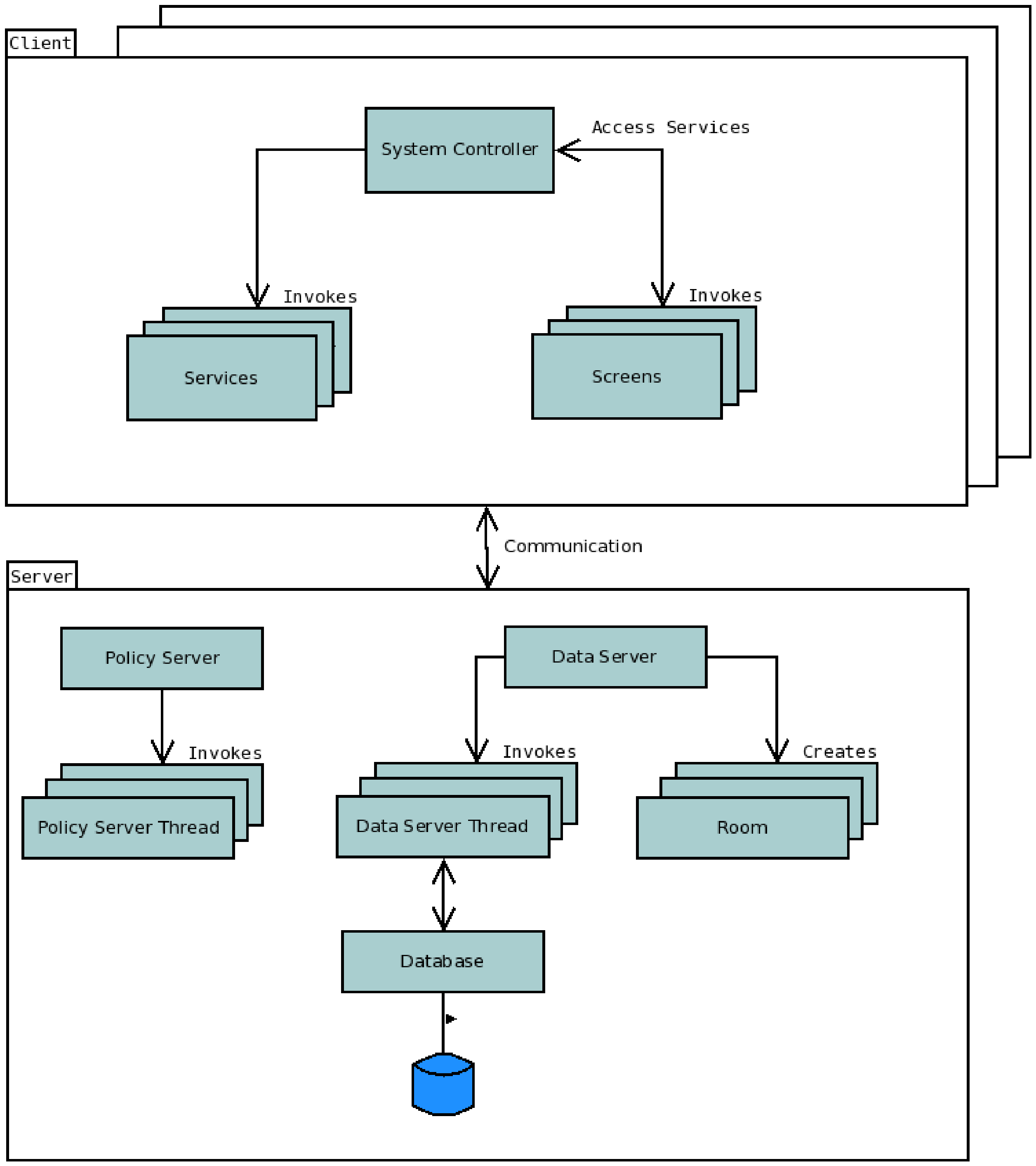
2. REFERENCES

[None]

3. DECOMPOSITION DESCRIPTION

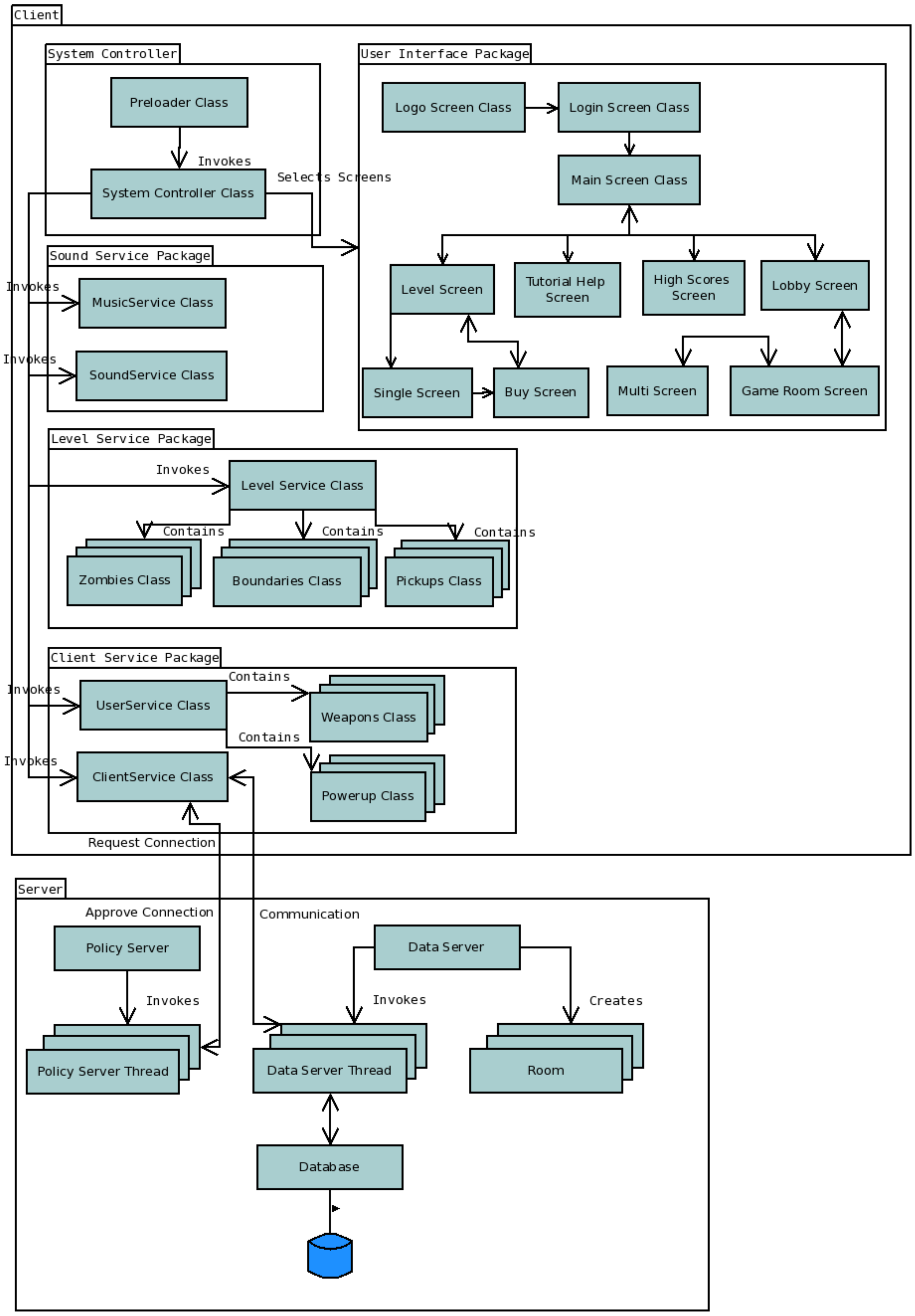
3.1 MODULE DECOMPOSITION

Zombie Apocalypse uses a client-server architecture. The client contains numerous sub-systems that are very modular (Screens), and a few sub-systems that provide access to data (Services). This increases component cohesion and decreases component coupling. In general, the architecture of the game is:



More specifically, the interactions between the screens and services is more complex. Each one of the screen classes is completely decoupled from the other screens; they are connected together only by making a call to the system controller to switch the currently displayed screen. Furthermore, each screen's visual information is separated from the controlling logic. The visual information is edited in Adobe Flash and stored as an .swc file. The controlling logic is written in the screen class. When each screen is instantiated, it loads its appropriate .swc file. If a screen requires access to certain data, it will load the appropriate services from the SystemController. Screens use the services to access data, edit data, and communicate with the server. In this way, the view (.swc file) and model (data inside UserService, LevelService, etc) are separated by the controlling classes (screens).

A more detailed diagram of the game architecture is described here:

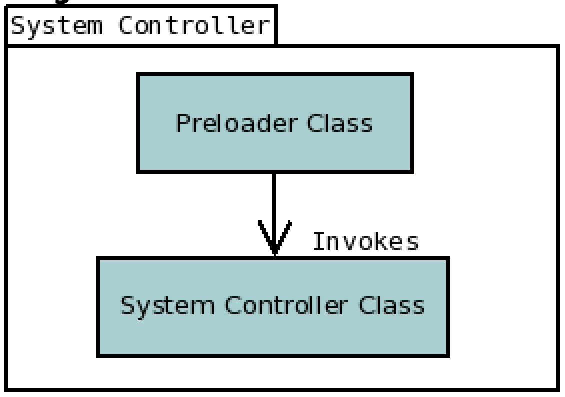

Client
System Controller
Preloader Class
Invokes
System Controller Class
Selects Screens
User Interface Package
Logo Screen Class
Login Screen Class
Main Screen Class
Level Screen
Tutorial Help Screen
High Scores Screen
Lobby Screen
Single Screen
Buy Screen
Multi Screen
Game Room Screen
Sound Service Package
Invokes
MusicService Class
Invokes
SoundService Class
Level Service Package
Invokes
Level Service Class
Contains
Contains
Contains
Zombies Class
Boundaries Class
Pickups Class
Client Service Package
Contains
Invokes
UserService Class
Weapons Class
Contains
Invokes
ClientService Class
Powerup Class
Request Connection
Server
Approve Connection
Policy Server
Invokes
Policy Server Thread
Communication
Data Server
Invokes
Creates
Data Server Thread
Room
Database

3.1.1 System Controller

Author: Chad Nelson

Description: This is the system that bootstraps the project. It initializes services and screens and then displays the team logo.

Diagram:



Components:

- **Preloader Class** - Initialized while the user is downloading the executable. It displays the progress of the download, and when complete, boots the SystemController
- **SystemController Class** - Bootstraps the game. Initializes all services and screens and displays the LogoScreen.

Services:

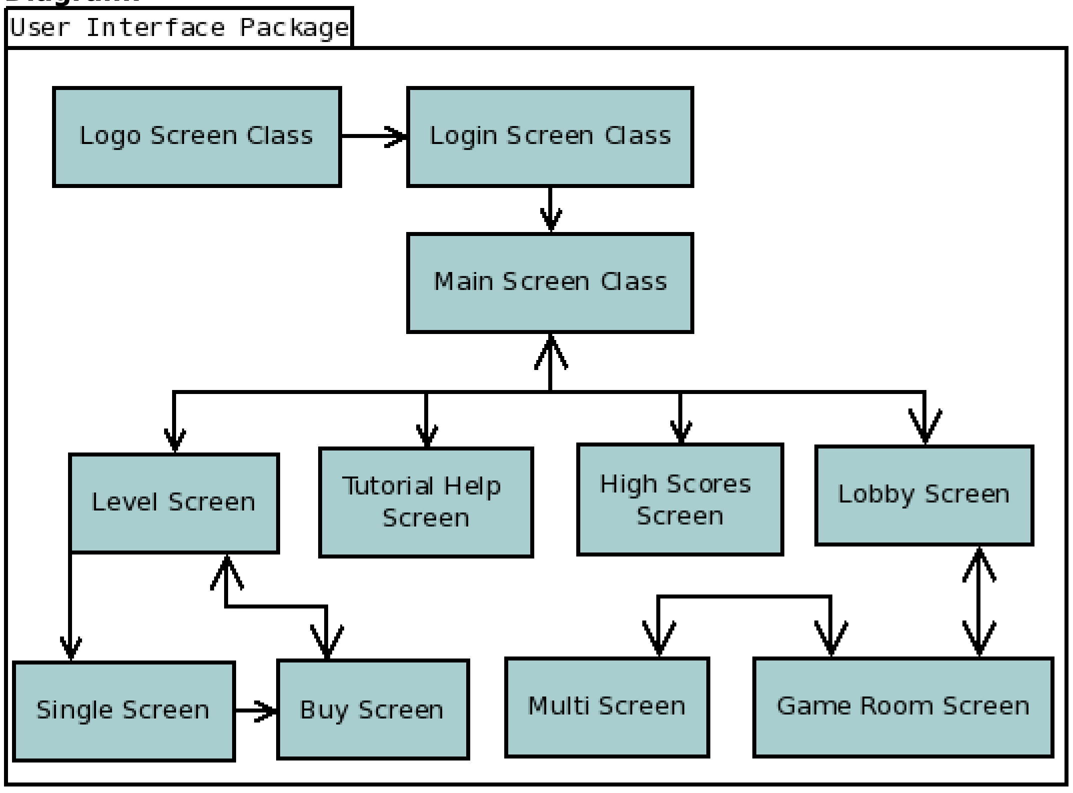
- Holds references to services and screens
- Enables switching screens of the User Interface Package
- Initializes Screens and Services

3.1.2 User Interface Package

Author: Cole Anagnost

Description: The User Interface Package is responsible for allowing the user to interact with the client, and processes all user input. This package includes the screens that make up the game's main menu, as well as the displays for both single and multi-player gameplay.

Diagram:



Components:

- **BuyScreen Class** - Allows the player to purchase new weapons or ammunition for weapons they already own.
- **HelpScreen Class** - Displays the game controls.
- **LevelScreen Class** - Allows the player to pick a level in the single player campaign.
- **LobbyScreen Class** - Displays the multi player chat lobby, and allows the player to select a game to join.
- **LoginScreen Class** - Allows the player to enter their usename and password, then verifies those with the server.
- **LogoScreen Class** - Displays the game logo while the client connects to the server.
- **MainScreen Class** - Displays the main menu screen.
- **MultiScreen Class** - Displays the multi player game screen (level plus user interface overlay, plus other players). Receives user input during gameplay.

- **RoomScreen Class** - Displays the multi player game room. Allows the player to chat with other players in the game room, and allows the host to change game settings.
- **ScoreScreen Class** - Displays the top scores for each level, retrieved from the server.
- **SingleScreen Class** - Displays the single player game screen (level plus user interface overlay). Receives user input during gameplay.

Services:

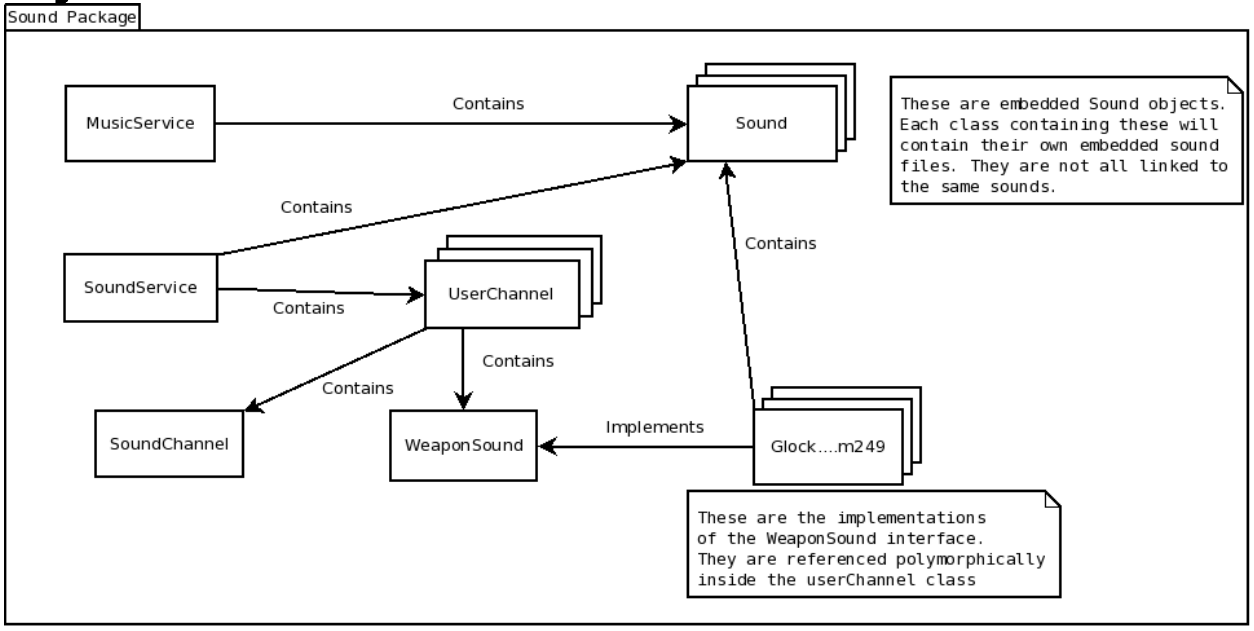
- Provides a menu interface for the user upon login (via MainScreen)
- Provides an interface for purchasing new weapons or ammunition (via BuyScreen)
- Displays game controls (via HelpScreen)
- Provides an interface for selecting a level in the single player campaign (via LevelScreen)
- Provides an interface for chatting with other players (via LobbyScreen and RoomScreen)
- Provides an interface for creating new game rooms (via LobbyScreen)
- Provides an interface for the user to enter their username and password (via LoginScreen)
- Provides an interface for the game host to change game settings (via RoomScreen)
- Displays the top scores retrieved from the server (via ScoreScreen)
- Provides an in-game interface overlay for the user(displaying player vitals such as health and ammo)
- Receives and processes user input during gameplay (via SingleScreen and MultiScreen)

3.1.3 Sound Service Package

Author: Tasewell Fox

Description: The Sound Package controls and regulates all sound occurring in the game. It is responsible for playing sound effects, music and keeping the sound data organized. The sound package consists of two major classes, the SoundService and MusicService, which provide the sound capabilities. The SoundService class plays all of the sound effects of the game both local and remotely, and the MusicService class plays sound locally.

Diagram:


Sound Package
MusicService
Contains
Sound
These are embedded Sound objects. Each class containing these will contain their own embedded sound files. They are not all linked to the same sounds.
Contains
Contains
SoundService
UserChannel
Contains
Contains
Contains
SoundChannel
WeaponSound
Implements
Glock....m249
These are the implementations of the WeaponSound interface. They are referenced polymorphically inside the userChannel class

Components:

- **MusicService Class** - The MusicService class is responsible for controlling the play back of music.
- **SoundService Class** - The SoundService class is responsible for playing all of the sound effects from the game.
- **WeaponSound Interface -** The WeaponSound interface provides a standard interface for all of the sound effects of the game.
- **UserChannel Class -** The UserChannel class creates and stores sound channels for each player to use so multiple sounds can be played concurrently.
- **Sound Class** - This is the Flex 3 standard library sound class.
- **SoundChannel Class** - This is the Flex 3 standard library soundChannel class.

Services:

- Provides music for the client side
- Provides the client with functions to select, start, and stop music
- Provides weapon firing sound effects
- Provides weapon reloading sound effects
- Provides weapon deployment sound effects
- Provides weapon switching sound effects

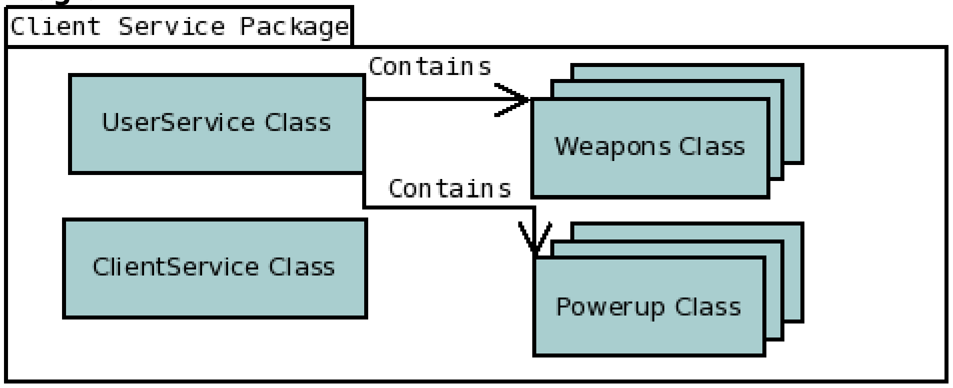
- Provides the ability to play sound effects on specific channels depending on which user has initialized the sound event

3.1.4 Client Service Package

Author: Tasewell Fox

Description: The Client Package provides the local client with mechanisms to store, aggregate and manipulate data. This package is responsible for creating a socket connection with the server, decoding server commands, sending data to the server, data logging, and debugging.

Diagram:



Components:

- **ClientService Class** - Responsible for creating a socket connection with the game server, sending messages to the server, and handling incoming server messages.
- **UserService Class** - Holds user data and communicates updates to the user data to ClientService

Services:

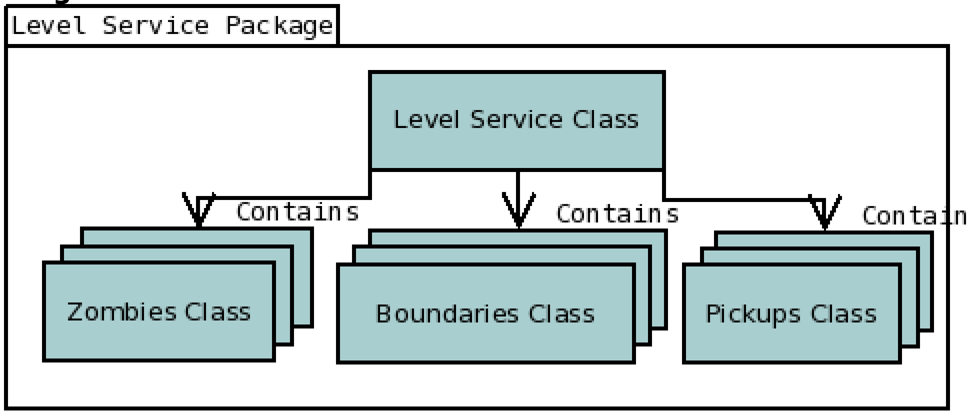
- Creates connection to the server
- Provides user authentication to server
- Receives and stores user data from server
- Sends updates of user data to server

3.1.5 Level Service Package

Author: Tim Flanigan

Description: This package provides a single service and a set of objects that are used during gameplay. Only the SingleScreen class and the MultiScreen class use the LevelService.

Diagram:



Components:

- **LevelService class -** This class is responsible for creating and holding all the level data. Levelservice creates a new level that contains randomly placed zombies inside certain boundaries that are also stored in the class.
- **Pickups Class -** This class is responsible for any pickups that get dropped when the user kills a zombie. Depending on what pickup is obtained, a pickup allows the user to increase their health, money, or ammo.
- **Boundary Class -** This class is responsible for handling player and zombie movements that take place near a boundary. It keeps users and zombies from moving into boundary that is not allowed.
- **Zombie class -** This class stores all information about zombies including health, speed, movement data. This class is responsible for the various ways that a zombie can move towards a player.

Services:

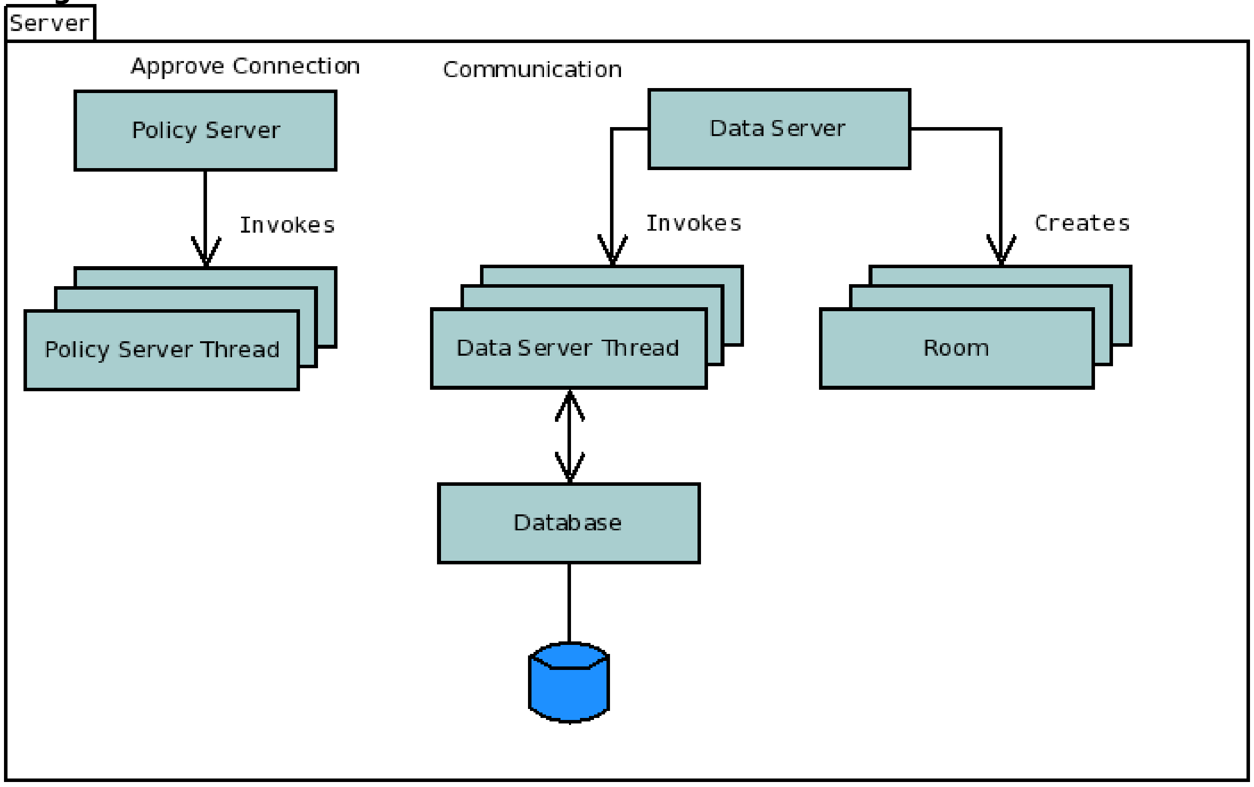
- Creates a new level
- Spawns zombies randomly inside valid points on the level.
- Sets difficulty of zombies spawned
- Contains data on the invalid areas a user or zombie can move to inside the level.
- Zombie class controls zombie movement throughout the level.
- Displays pickups on the level after a zombie is killed if one drops.

3.1.6 Server Package

Author: Chad Nelson

Description: The server package allows persistent user data and multiplayer communication. Part of Adobe's cross-domain security policy requires us to host a Policy Server that allows clients access to the Data Server port.

Diagram:



Components:

- **PolicyServer Class** - A socket server to handle incoming socket connections
- **PolicyServerThread Class** - A thread to handle policy requests from a client.
- **DataServer Class** - A socket server to handle incoming socket connections
- **DataServerThread Class** - A thread to handle incoming messages from a client.
- **Database Class** - This class establishes a connection to the database server and creates a table that contains the information of each users accomplishments. It also contains methods to query the server.
- **Room Class** - A collection of users

Services:

- Grant the client access to the DataServer port
- Allow the client to login / create a new user
- Sends the client its user data upon login
- Allow a logged in client to send chat messages and multiplayer commands to other sockets in the same room

- Allow a logged in client to retrieve an ordered list of high scores
- Allow a logged in client to create new rooms / join existing rooms
- Allow a logged in client to update their stats in the database

3.2 CONCURRENT PROCESS

3.2.1 Server Process

Name: DataServer

Description: The server side process for the game. Provides communication between game clients and the database.

Created: N/A

Terminated: N/A

Threads:
- DataServerThread - Handles all incoming messages from a particular client (one thread for each connected client)

3.2.2 Client Process

Name: Preloader / SystemController

Description: The client process contains the primary thread for the client application.

Created: This process is created when the client starts.

Terminated: This process is terminated when the client is terminated.

Threads:
- None (the only threading is hidden from us in ClientService)

3.3 DATA DECOMPOSITION

3.3.1 Server Database

Column Name	Type	Description
username	TEXT(50)	Player's username
password	TEXT(50)	Player's password
score	INT	Player's score (used to rank)
level	INT	Player's highest unlocked level
wins	INT	Number of levels completed
losses	INT	Number of failed attempts
cash	INT	Player's current cash

kill_count	INT	Player's total kills
accuracy	INT	Player's overall accuracy rating
weapons	TEXT(500)	Player's current weapon and ammo list

3.3.2 LevelService

Description: Represents the current level.

Fields:

public var widget : MovieClip
Background image of the current level

public var levelIndex : int
Current level

public var users : Array = new Array()
Array of UserService objects (each player in the game)

public var boundaries : Array
Array of Boundary objects

public var zombies : Array = new Array()
Array of Zombie objects

public var pickups : Array
Array of Pickup objects that zombies drop

public var difficulty : int
Difficulty of zombies (0 - 5)

public var width : Number
Width of the level widget

public var height : Number
Height of the level widget

public var numZombies : int
Number of zombies in level (used for stat tracking)

public var bloodsplatters : MovieClip
Array for holding blood splatters

3.3.3 Weapon Class

The weapon class contains information pertaining to all weapons in the game. It contains all the statistics for the weapon.

Fields:

public var name : String;
 Name of the weapon

public var graphic : DisplayObject;
 Graphic object that displays the weapon

public var frames_per_bullet : int;
 Number of non-shooting frames that pass before next shot

public var frames_per_reload : Number;
 Number of frames before gun is reloaded

public var penetration : int;
 Number of zombies the round will penetrate through

public var spread : int;
 The angle in degrees that the gun might fire off center

public var range : int;
 How far the bullet will travel

public var ammoLoaded : int;
 Ammo currently in the gun

public var ammoLeft : int;
 Total ammo remaining

public var damage : int;
 Damage the gun will do

public var clipSize : int;
 The number of bullets in a clip

public var sound : int;
 Range that sound attracts attention

public var knockback : int;
 How far the gun knocks back its target

public var velocity : Number;
 Velocity of the round.

public var shots : int;
 The number of shots discharged each time the weapon is fired.

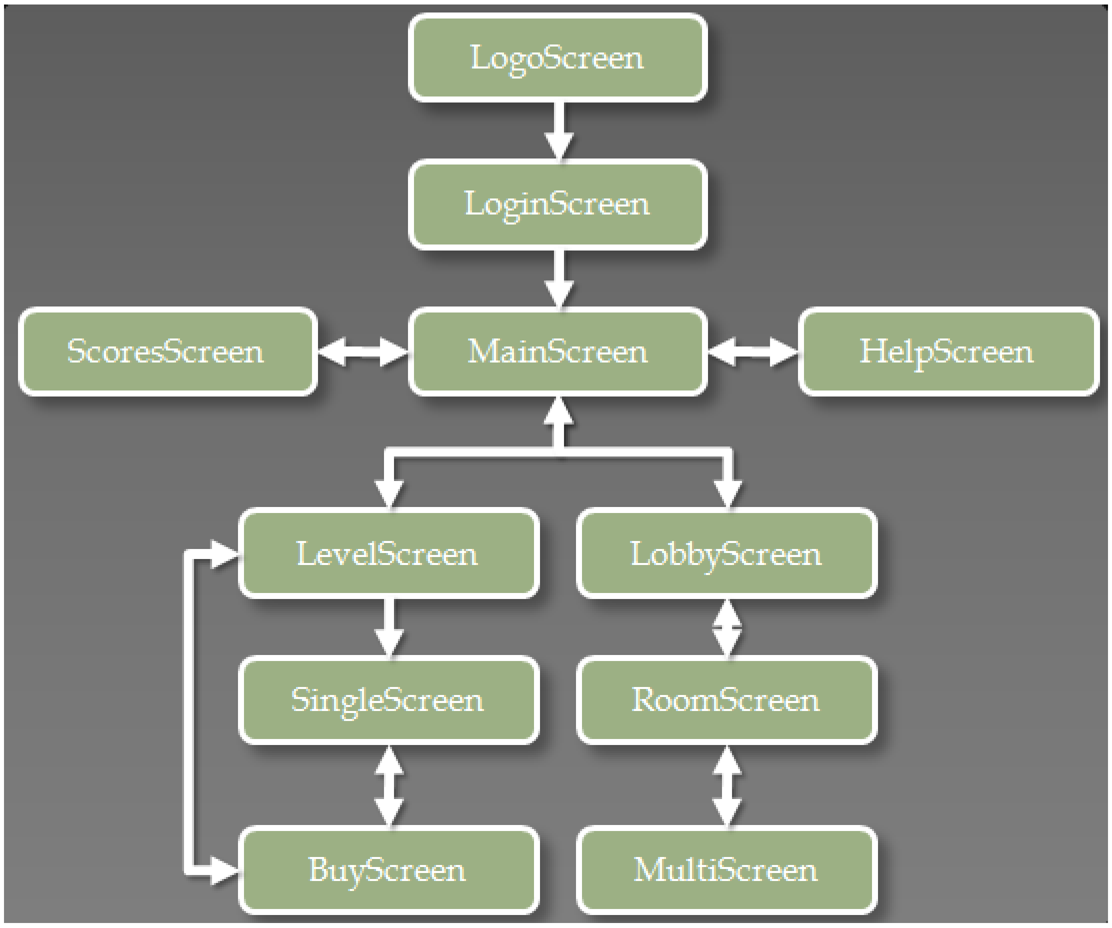
public var cost : int;
 Cost of the weapon in the store

public var primary : Boolean;
 True for primary, false for secondary

public var type : int;
 Index value from WeaponIndex

3.4 STATES

3.4.1 Game State

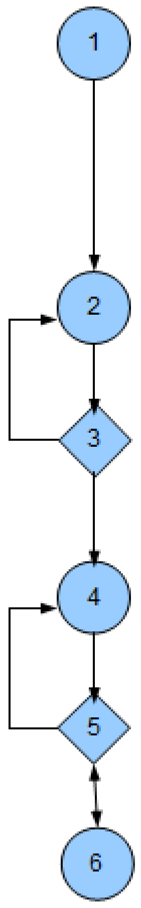


States:

LogoScreen: Switches to LoginScreen after showing the title screen
LoginScreen: Switches to MainScreen after user enters their username/password
MainScreen: Switches to ScoresScreen, HelpScreen, LevelScreen or LobbyScreen depending on user selection
ScoresScreen: Returns to MainScreen on user selection
HelpScreen: Returns to MainScreen on user selection
LevelScreen: Switches to MainScreen or SingleScreen depending on user selection
LobbyScreen: Switches to MainScreen or RoomScreen depending on user selection
SingleScreen: Switches to BuyScreen at the end of a level
RoomScreen: Switches to LobbyScreen or MultiScreen depending on user selection
BuyScreen: Returns to SingleScreen or switches to LevelScreen depending on user selection

MultiScreen: Returns to RoomScreen at the end of a level

3.4.2 Zombie State

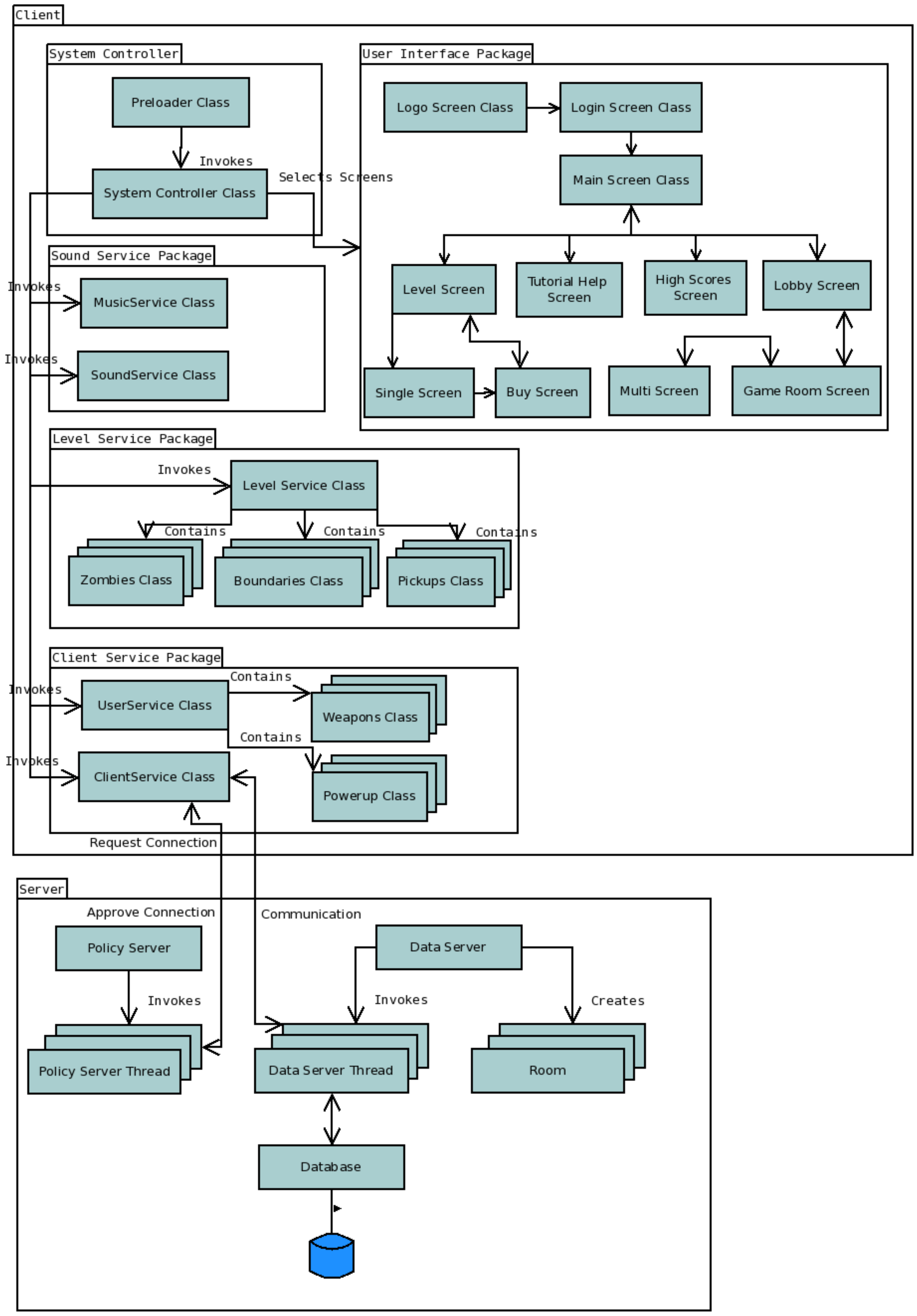


States:

1. Zombie Initialization: In this state the zombies are initialized
2. Random Walk: The zombie randomly selects a target and walks to it
3. Is a player detected?: Checks if the zombie has detected a player
4. Zombie chase player: Zombie goes after the player
5. Zombie in attack distance?: Checks if the zombie is in attack range
6. Zombie attack: Zombie attacks the player

4. DEPENDENCY DESCRIPTION

4.1 INTERMODULE DEPENDENCIES


Client
System Controller
Preloader Class
Invokes
System Controller Class
Selects Screens
User Interface Package
Logo Screen Class
Login Screen Class
Main Screen Class
Level Screen
Tutorial Help Screen
High Scores Screen
Lobby Screen
Single Screen
Buy Screen
Multi Screen
Game Room Screen
Sound Service Package
Invokes
MusicService Class
Invokes
SoundService Class
Level Service Package
Invokes
Level Service Class
Contains
Contains
Contains
Zombies Class
Boundaries Class
Pickups Class
Client Service Package
Invokes
UserService Class
Contains
Weapons Class
Contains
Invokes
ClientService Class
Powerup Class
Request Connection
Server
Approve Connection
Policy Server
Invokes
Policy Server Thread
Communication
Data Server
Invokes
Creates
Data Server Thread
Room
Database

4.2 INTERPROCESS DEPENDENCIES

NONE

4.3 DATA DEPENDENCIES

See Intermodule dependencies

5. INTERFACE DESCRIPTION

5.1 MODULE INTERFACE

5.1.1 SystemController

Description: The SystemController is a singleton class that holds references to services and screens, and provides public methods to change the currently displayed screen, and to get the singleton instance of the SystemController.

Public Functions
getInstance():SystemController
changeScreen(previous : DisplayObject, index:int):void
fadeChangeScreen(previous : DisplayObject, index : int):void
animateChangeScreen(previous : DisplayObject, index : int, reverse : Boolean = false):void

The index value passed to changeScreen functions is specified by the enumerator ScreenIndex. For example, if the LogoScreen is done displaying the logo and wants the Login screen to load, it will call:

SystemController.getInstance().animateChangeScreen(this, ScreenIndex.LOGIN);

Public Variables	Description
clientService : ClientService	Reference to the ClientService
levelService : LevelService	Reference to the LevelService
musicService : MusicService	Reference to the MusicService
soundService : SoundService	Reference to the SoundService
userService : UserService	Reference to the UserService
screens : Array	Untyped array of all screens from the User Interface Package
singlePlayerMode : Boolean	A read-only boolean indicating game mode

5.1.2 ClientService

Description: The main communications interface is the boundary between client and server. Although the level of detail covered below is more fit for a design document, we list it here for transparency.

Our server will always be running and listening for incoming socket connections on the following DNS and port:

Host = chad.game-host.org
Port = 25432

Once a socket connection has been opened, UTF string messages will be sent between the client and server based on the criteria outlined in this section.

Messages are formatted by having an operation string followed by a colon, with any parameters following the opcode. Multiple parameters will be delimited by a single "|" character.

The format to describe our communications interface in the below scenarios is
>> Opcode: - text description of the opcode (param1, param2, ...)
where ">>" indicates a message the client sends to the server and "<<" indicates a message a message sent from the server to the client. "><" indicates a message that is sent both ways.

Public Functions
socketSend(message:String):void

Opcode (Sending)	Arguments	Description
LOGIN	username|password	Logs in the user or returns an error
CREATEUSER	username|password	Creates a new user or returns an error
CHECKUSER	username	Returns USERAVAILABLE if this username is not used; nothing otherwise
JOINROOM	roomname	Makes the user join the room, leaving his old room
CHAT	message	Sends a chat to the room the user is in
GETHIGHSCORES		Retreive high scores from the server
UPDATEUSERVAR	variable|value	Updates a column in the database for this user
S*	Multiplayer setup commands	A special chat-like command used for setting up multiplayer
M*	Multiplayer commands	A special chat-like command used in multiplayer

Opcode (Receiving)	Arguments	Description
LOGIN	username=bobdylan|score=...	User data sent after log in / user creation

LOGINFAILED	errormsg	Error message if login failed
CREATEUSERFAILED	errormsg	Error message if user creation failed
USERAVAILABLE	username	Username is available for user creation
HIGHSCORES	1,chad,100,...|r2c1,r2c2,r2c3...	High scores list
LOBBYUSERLIST	user1|user2|user3...	List of users in the lobby
LOBBYROOMLIST	room1|room2|room3...	List of available rooms
LOBBYUSERJOIN	username	Presence notification: user joined lobby
LOBBYUSERLEFT	username	Presence notification: user left lobby
CREATEROOM	roomname	Room presence notification
DELETEROOM	roomname	Room presence notification
LOBBYCHAT	message	Chat message from a user in the lobby
USERLIST	user1|user2|user3...	List of users in the game room
USERJOIN	username	Presence notification: user joined room
USERLEFT	username	Presence notification: user left game room
CHAT	message	Chat message from a user in a game room
S*	args	Multiplayer Setup Command
M*	args	Multiplayer Command

The class also has one public variable:

Public Variables	Description
connected:Boolean = false;	A read-only boolean indicating connection status

5.1.3 LevelService

Description: The LevelService is responsible for maintaining the graphical representation of the currently played level.

Public Functions	Description
setLevel(index : int):void	Erases the current level and constructs the start of level index

Public Variables	Description
widget:MovieClip	Map image
levelIndex:int	Is this level 1, level 2, etc?
boundaries:Array	Array of Boundary objects that no one can enter

zombies:Array	Array of Zombie objects
pickups:Array	Array of Pickup objects that zombies drop
bloodsplatters:MovieClip	Movieclip to contain bloodsplatters
difficulty:int	Difficulty of zombies
width:int	initial width of the widget
height:int	initial height of the widget
numZombies:int	Number of zombies on map at the start of the level

5.1.4 MusicService

Description: The music service contains two public functions. It pulls its music from the internet by containing URLs to .mp3 files.

Public Functions	Description
play(levelIndex : int):void	Starts playing the music for the level specified
fadeOut():void	Fades out the music

5.1.5 SoundService

Description: The sound service is responsible for playing all of the sound effects from the game. This class has three public functions which control how the sounds are played and on what channel. When a user joins the game they are assigned a sound channel using the addUser function. When a player takes an action that would play a sound a call to the sound service is issued using the localCall function. An opcode string is passed to the sound service decoded and played on the appropriate channel, and then forwarded to the server so other players can also hear the sound.

Public Functions	Description
localCall(args:String):void	Makes a local sound call and pushes a sound event to the server
ServerCall(args:String):void	A sound call from the server, plays the sound on the appropriate sound channel.
addUser(name:String, weapon:int):void	This function adds a user channel object to the soundService initialized with the username and the index of the users current weapon

5.1.6 UserService

Description: The user service is a data storage class that stores all of the information of the current player and pushes it to the server when the player is done.

Public Functions	Description
setup(args:String):void	This function takes a string containing all of the stored users data and initializes the user service with the data.

syncAll():void	Pushes all of the modified user data back to the server for storage.
tallyShot(w:Weapon):void	Keeps a running count of the number of times a weapon is fired, used for stat tracking
tallyBullet(b:Bullet):void	Keeps a running count of bullets that hit their target.

Public Variables	Description
name:String	The players name
money:int	The amount of money the player has
level:int	The level the player is on
kill_count:int	Running count of the number of kills
hit_count:int	number of bullets that hit
shot_count:int	number of bullets shot
win_count:int	number of wins
loss_count:int	number of losses
score:int	the players current score
health:int	the players current healt
weaponPrimary:Weapon	The current primary weapon
weaponSecondary:Weapon	The current secondary weapon
powerup:Powerup	The current powerup
weapons:Array	All weapons owned by the player
powerups:Array	All powerups owned by the player
items:Array	All items owned by the player
widget:w_player	The players graphical widget
velocity:Point	The players current velocity
speed:int	The players current speed

5.1.7 BuyScreen

Description: The buyscreen is displayed after every level in the game. It allows the user to choose weapons and equipment to use in the game. Any weapons or equipment the player buys through this screen is stored in an array in the UserSerivce.

Public Functions	Description
setMessage(I_STATUS: int):void	Sets the continue button message

Public Constants	Description
I_FAILED : int	Int used to change the continue button message
I_SUCCESS : int	Int used to change the continue button message
I_LEVEL : int	Int used to know which level the continue button goes too

5.1.8 LobbyScreen

Description: The lobby screen is displayed when the user selects "multiplayer" from the main menu. It allows the user to chat with other users and join or create multiplayer games.

Public Functions	Description
setUserList(users : String):void	Receives the lobby user list from the server
setRoomList(rooms : String):void	Receives the lobby room list from the server
userJoin(username : String):void	A user has joined the lobby (add them to the user list
userLeft(username : String):void	A user has left the lobby (remove them from the user list)
roomCreated(roomname : String):void	A room has been created (add it to the room list)
roomDeleted(roomname : String):void	A room has been deleted (remove it from the room list)
receiveChat(msg : String):void	Receives a chat message from the server
gotoGameRoom():void	Takes user to the game room screen

5.1.9 LoginScreen

Description: The login screen takes a username and password that the user inputs and sends it to the server to check if it's valid. If the information is valid, the user logs into there account, otherwise an error message is displayed.

Public Functions	Description
loginFailed(msg:String):void	Displays a message saying login failed if something goes wrong
createUserFailed(msg:String):void	Displays a message saying unable to create user if the account could not be created
userAvailable(username:String):void	Changes the available variable if the username was available

5.1.10 MultiScreen

Description: The MultiScreen extends SingleScreen, overwritting only a few of it's methods.

Public Functions	Description
receiveChat(msg : String):void	Receive a chat message during gameplay

5.1.11 RoomScreen

Description: The room screen is displayed when the user joins a multiplayer game from the lobby screen. It allows the user to chat with other users in the game room, and allows the host to change game settings.

Public Functions	Description
setUserList(users : String):void	Receives game room user list from the server
userJoin(username : String):void	A user has joined the game room (add them to the user list)
userLeft(username : String):void	A user has left the game room (remove them from the user list)
receiveChat(msg : String):void	Receives a chat message from the server
receiveSetProperty(cmd : String, value : String):void	Receives a change in game settings made by the host

5.1.12 ScoreScreen

Description: This class represents the High score screen in the game. It reads data from the database and displays the users ranked by score on the screen.

Public Functions	Description
loadScores(highscores:String):void	Reads user data from the database and displays it on screen in ranked order

5.1.13 ScreenIndex

Description: An enumerator for the different screens in the game

Public Constants
BUY:int = 0
HELP:int = 1
LEVEL:int = 2
LOBBY:int = 3
LOGIN:int = 4
LOGO:int = 5
MAIN:int = 6
MULTI:int = 7
ROOM:int = 8
SCORE:int = 9
SINGLE:int = 10

5.1.14 Bullet

Description: This class represents a bullet depending on what type the weapon using the bullet shoots. It is represented graphically as well as by the data the object holds.

Public Functions	
Bullets(gun : Weapon, shape : Shape, tweenP : TweenLite, target : Point)	Creates a bullet object as the bullet is being shot from the gun
teardown():void	Removes the bullet from the screen

Public Variables	Description
graphic : Shape	Represents the shape of the bullet depending on the ammo and gun type
tween : TweenLite	
penetration : int	Represents the number of zombies the bullet can penetrate through when shot
endPoint : Point	Represents the point where the bullet dissapears
lifeTime : int	Represents the time the bullet can be displayed if it doesn't hit anything
weapon : Weapon	Represents the weapon the bullet is being used for
hit : boolean	Used for stat tracking if a bullet hits a target

5.1.15 Boundary

Description: A Boundary is an object that contains two points and a type. It represents a two-dimensional shape in a level that is a zone of no entry.

Public Functions	Description
calculatePosition(initialPosition:Point, desiredPosition:Point):Point	Determines the location of the traveling object after calculating collisions with this boundary
checkCollision(initialPosition:Point, desiredPosition:Point):Boolean	Does a quick check to see if the traveling object intersects this boundary

5.1.16 BoundaryIndex

Description: An enumerator for boundary type objects.

Public Constants
FENCE : int = 0
RECTANGLE : int = 1
CIRCLE : int = 2

5.1.17 Pickup

Description: This class is responsible for any pickups that get dropped when the user kills a zombie. Depending on what pickup is obtained, a pickup allows the user to increase their health, money or ammo.

Public Functions	Description
Pickup(index : int, value : int)	Displays a pickup item on the screen when needed

Public Variables	Description
widget : DisplayObject	The pickup graphic
health : int	The amount of health the pickup will give the user when picked up
cash : int	The amount of cash the pickup will give the user when picked up
ammo_type : int	The type of ammo the pickup represents
ammo : int	The amount of ammo the pickup will give the user when picked up
lifetime : int	The time the pickup will stay on the screen before it dissapears

5.1.18 PickupIndex

Description: An enumerator for available pickup items

Public Constants
CASH : int = 0
HEALTH : int = 1
PRIMARYAMMO : int = 2
SECONDARYAMMO : int = 3

5.1.19 Powerup

Description: This class is responsible for any powerups dropped when the user kills a zombie. A powerup gives the user a temporary bonus when collected.

Public Functions	Description
Powerup(index : int)	Displays a powerup item on the screen when needed

Public Variables	Description
widget : DisplayObject	The powerup graphic

index : int	The type of powerup (based on the PowerupIndex enumerator)
charges : int	The number of times the powerup can be used before it runs out

5.1.20 PowerupIndex

Descrption: An enumerator for available powerup items

Public Constants
BODYARMOR : int = 0
MEDKIT : int = 1
MAXAMMO : int = 2
BOMB : int = 3

5.1.21 Weapon

Description: This class is responsible for storing data for weapons in the game.

Public Functions	Description
fire() : void	Plays the sound for a weapon firing
reload() : void	Starts a weapon reload (plays sound and delays call to reloadWeapon)
empty() : void	Plays the sound for an empty weapon firing
toString() : String	Displays weapon info in a human readable format
toServerString() : String	Creates a string to send to the server with weapon data

Public Variables	Description
name : String	Name of the weapon
graphic : DisplayObject	The weapon graphic
frames_per_reload : Number	The number of frames it takes to reload the weapon
penetration : int	The number of enemies the bullet will penetrate
spread : int	The maximum angle in degrees that the gun may fire off center
range : int	The distance the bullet will travel
ammoLoaded : int	The amount of ammo currently in the weapon

ammoLeft : int	The total ammo the player has remaining for this weapon
damage : int	The damage done per round
clipSize : int	The number of rounds in one clip
sound : int	The range at which firing this weapon will attract zombie attention
knockback : int	The distance the weapon will knock back a target
velocity : Number	The speed of the bullet
shots : int	The number of shots discharged each time the weapon is fired
cost : int	The price of the weapon in the store
primary : Boolean	True for primary, false for secondary
type : int	The weapon's WeaponIndex
reloading : Boolean = false	Whether or not the weapon is currently reloading

5.1.22 WeaponIndex

Description: An enumerator for the weapons in the game

Public Constants
PISTOL : int = 0
MAGNUM : int = 1
MAC10 : int = 2
MP5 : int = 3
SHOTGUN : int = 4
AUTOSHOTGUN : int = 5
M4A1 : int = 6
AK47 : int = 7
SNIPERRIFLE : int = 8
MAGNUMSNIPER : int = 9
MACHINEGUN : int = 10
number_of_weapons : int = 11

5.1.23 Zombie

Description: This class is responsible for storing the data and controlling the behavior of the zombies. This class contains methods to update the zombies AI routines and their movements.

Public Functions	Description
refresh(player:UserService):void	This function refreshes the zombies AI against a player. If the zombie is not delayed and within range it will attack the player.
move(pos:Point):void	Moves the zombie one unit towards the point passed to it. This function also rotates the zombie so they are facing that point.
moveAngle(pos:Point, angle:Number):void	Moves the zombie one unit towards the point passed to it. This function rotates the zombie by the angle passed to it.
shot(player:UserService,gun:Weapon):void	This function is called when the zombie is shot to calculate damage and apply the knockback vector.
dropPickup():void	This function is responsible for randomly determining if a pickup item is dropped when the zombie dies.

Public Variables	Description
graphic : w_zombie	The zombies graphic
health : int	The zombies current health
velocity : int	The zombies velocity
runVel : int	The zombies velocity when running
hitVel : int	The zombies velocity when shot
detected : Boolean	has the zombie detected a player?
hit : Boolean	has the zombie been shot?
distance : Number	The distance between the player and the zombie
traveled : int	The distance the zombie has traveled for pathing purposes
target : Point	The zombies target for pathing purposes
type : int	The type of zombie, runner = 1, zig-zag = 2
PauseTime : int	Pausing variable for the random walk pathing
delayTime : int	Zombies delay time after hitting the player
damage : int	The amount of damage the zombie does

5.1.24 ZombieIndex

Description: An enumerator for the zombie types in the game

Public Constants
SLOW : int = 0
FAST : int = 1
ZIGZAG : int = 2

5.1.25 PolicyServerThread

Description: The PolicyServer is responsible for conforming to Adobe's Cross-Domain Security policy. If we want an instant socket connection, then we are to implement a policy server on port 843 that listens for policy file requests, and returns the policy file. Thus, our socket server accepts incoming connections, listens for a specific xml string, and returns an xml policy file that allows cross domain access to our dataserver on port 25432

Policy File Request:

```
<policy-file-request/>
```

Policy File Response:

```
<?xml version=\"1.0\"?>
<!DOCTYPE cross-domain-policy SYSTEM \"http://www.adobe.com/xml/dtds/cross-domain-policy.dtd\">
<cross-domain-policy>
    <allow-access-from domain=\"*\" to-ports=\"25432\" />
</cross-domain-policy>
```

5.1.27 DataServer

Description: The DataServer is responsible for spawning DataServerThreads when new incoming sockets connect. It is also a place that holds shared data, such as usernames and rooms.

Public Functions	Description
public void sendToRoom(String message, String socket)	Sends message to users in the room
public void sendToSocket(String message, Socket socket)	Sends a message to a user

Public Variables	Description
Hashtable<Socket, String> usernames	List of logged inusernames and their sockets
Hashtable<Socket, Room> rooms	List of sockets and their associated Room object
Hashtable<String, Room> roomnames	List of roomnames and their associated Room object

5.1.27 DataServerThread

Description: The DataServerThread is created upon the dataserver accepting an incoming socket connection. The DataServerThread is responsible for handling all messages sent it from the ClientService. The messages it handles and sents are repeated here:

Opcode (Receiving)	Arguments	Description
LOGIN	username|password	Logs in the user or returns an error
CREATEUSER	username|password	Creates a new user or returns an error
CHECKUSER	username	Returns true if this username is not used

JOINROOM	roomname	Makes the user join the room, leaving his old room
CHAT	message	Sends a chat to the room the user is in
GETHIGHSCORES		Retreive high scores from the server
UPDATEUSERVAR	variable|value	Updates a column in the database for this user
S*	Multiplayer setup commands	A special chat-like command used for setting up multiplayer
M*	Multiplayer commands	A special chat-like command used in multiplayer

Opcode (Sending)	Arguments	Description
LOGIN	username=bobdylan|score=...	User data sent after log in / user creation
LOGINFAILED	errormsg	Error message if login failed
CREATEUSERFAILED	errormsg	Error message if user creation failed
USERAVAILABLE	username	Username is available for user creation
HIGHSCORES	1,chad,100,...|r2c1,r2c2,r2c3...	High scores list
LOBBYUSERLIST	user1|user2|user3...	List of users in the lobby
LOBBYROOMLIST	room1|room2|room3...	List of available rooms
LOBBYUSERJOIN	username	Presence notification: user joined lobby
LOBBYUSERLEFT	username	Presence notification: user left lobby
CREATEROOM	roomname	Room presence notification
DELETEROOM	roomname	Room presence notification
LOBBYCHAT	message	Chat message from a user in the lobby
USERLIST	user1|user2|user3...	List of users in the game room
USERJOIN	username	Presence notification: user joined room
USERLEFT	username	Presence notification: user left game room
CHAT	message	Chat message from a user in a game room
S*	args	Multiplayer Setup Command
M*	args	Multiplayer Command

5.1.28 Database

The server has one SQLite database with one table named "users." The table has the following columns and types:

Column Name	Type	Default Value	Example value	Meaning
username	TEXT(50)	None	bobdylan	Player's username
password	TEXT(50)	None	shotoflove	Player's password
score	INT	0	382	Player's score (used to rank)
level	INT	1	3	Level 3 is the highest unlocked level
wins	INT	0	43	Number of levels completed
losses	INT	0	12	Number of failed attempts
cash	INT	1000	3420	$3420
kill_count	INT	0	232	Player has killed 632 zombies
accuracy	INT	1000	934	93.4%
weapons	TEXT(500)	<0,12,96,0,0>	<0,12,96,934,1000><...>	<pistol,clip,ammo,hits,shots>
powerups	TEXT(50)		TBA	TBA

*Note that "username" is a primary key

5.1.29 Room

Description: The Room class is responsible for storing and manipulating the different multiplayer rooms that are created during the use of the program. It contains functions to add and delete users. Each user in the room is identified by their socket.

Public Functions	
void addUser(Socket socket)	This function adds a user to the room and notifies other members of the room to the arrival of the new user
void delUser(Socket socket)	This function removes a user from the room and notifies the other other members of the room of the departure of the user.

Public Variables	Description
String name	The name of the room
ArrayList<Socket> users	A list of the users sockets
long max	The maxmimum number of people that can be in the room

5.2 PROCESS INTERFACE

5.2.1 PolicyServer

Starting Process: The process is started via the command line. Only one PolicyServer process needs to exist.
Ending Process: This process is a long running process and should not end.

5.2.1.1 PolicyServerThread

Thread Start: A new thread is created for every new socket connection that is created. PolicyServer could hold on to any number of these worker threads.
Thread End: The thread terminates when the socket connection is terminated (the client on the other side is close or the connection is lost).

5.2.2 DataServer

Starting Process: The process is started via the command line. Only one DataServer process needs to exist.
Ending Process: This process is a long running process and should not end.

5.2.2.1 DataServerThread

Thread Start: A new thread is created for every new socket connection that is created. DataServer could hold on to any number of these worker threads.
Thread End: The thread terminates when the socket connection is terminated (the client on the other side is close or the connection is lost).

5.2.3 Client Program

Starting Process: The process can be started in multiple ways. One way is to run the .swf file using Adobe's Flash Player. Another way is to embed the .swf into a webpage and visit the page. Multiple clients can be spawned by the user.
Ending Process: This process ends when the user closes Adoble's Flash Player or leaves the webpage where the embedded .swf existed

*Note: This process contains no threads.

6. DETAILED DESIGN

NOT REQUIRED

7. DESIGN RATIONALE

7.1 DESIGN ISSUES

7.1.1 P2P vs. Client-Server

One of the major design decisions we had to make was whether to use a peer-to-peer (P2P) or client-server system for the network features of our program. We decided to use a client-server system. This decision was based on several factors, the most influencing being that the P2P features of actionscript are still in the beta stages and could not be guaranteed to be working 100%. If we could have had a more dependable P2P solution then we most likely would have implemented a UDP P2P connection instead of a TCP socket client-server model. The client server model also provides us with an easy way to integrate the sqlite database into the backend of the server so they could be tightly integrated.

7.1.2 Separate Multiplayer vs Extended Multiplayer

A very critical design decision we made late in the development of the program was deciding whether or not to develop an entirely separate multiplayer for our game or retool the single player class so that the multiplayer could extend it. The decision was made to retool the single player class mainly for the benefits of code re-use and simplicity. The ability to simply extend the single player gameplay elements allowed for a vast amount of code reuse within the program. The server opcodes could all remain the same between the single and multiplayer and only a few select function had to be overwritten and added to provide the functionality that was missing from the single player feature set. Extending the single player class was also a much more simple undertaking than attempting to derive a completely new class, and there's really no point in re-inventing the wheel.