Computer Science and Engineering, University of Nevada, Reno

BrainZ & GainZ

Team 14

Albert Wohletz, Brian Catudan, Collin Sorkin

Dr. Sergiu Dascalu

Dr. Sushil Louis

3/24/14

Abstract

Team 14's project will be to create a free to play mobile exploration game using the Unity3D engine. The game, BrainZ & GainZ, will have levels based on real world cities and will be modeled accurately to reflect a realistic but apocalyptic time. The purpose of the project is to introduce mobile gaming to a wider range of people by using the familiarity of levels to gain the interest of people who would normally not play mobile games. BrainZ & GainZ will be available for download through the Apple App Store and the Android Google Play Store.

Introduction

There are three main goals that the team would like BrainZ and GainZ to accomplish; the game will be playable with at least three levels upon release, receive at least 2,000 downloads between the Google Play Store and App Store, and introduce mobile gaming to a wider audience.

Understanding our users is key to developing a successful game. There are four main classifications of players detailed by the Bartle's player types: achievers, explorers, socializers, and killers. Achievers focus on winning in the game, explorers like to experience all a game has to offer, socializers like to interact with other players, and killers like to impose their will on other players (Despain). Team 14's intended users are casual gamers which we feel fits the achiever and explorer player types. Most people that play games on their phone are casual gamers to begin with, so by targeting those people the team feels that this will help us reach our goals. The way our team is going to achieve this is by creating controls that are very intuitive so that almost anyone can pick it up and play. By using the touch screen on a phone, users will easily be able to navigate the player throughout the levels.

All the main functionalities remain the same, as well as all of the team's goals for this game. The team has implemented enemy and power-up spawning. This allows the team to randomly spawn enemies or power-ups in random locations. There has also been an update in player movement that allows the player to move with the mouse instead of the keyboard. Using the teams newly implemented spawning functionality, the team is one step closer to finishing one complete playable level.

High-level Diagram

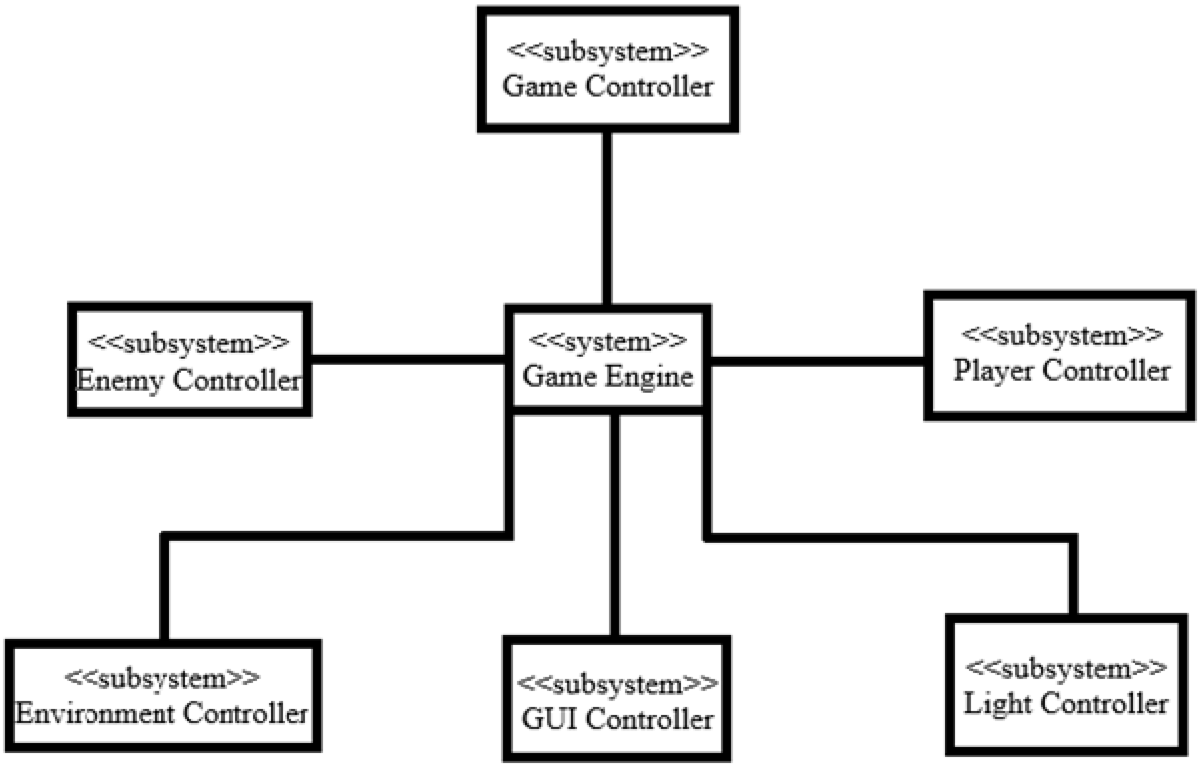


Figure 1: High-level diagram of BrainZ & GainZ

Game Controller
The Game Controller is a system that sits on top of the game engine and manages the game's state. This includes modules such as the Spawn Controller, which spawns game objects into the world.

Game Engine
The game engine is the core central component of the project that manages all of the controllers. In our project this is primarily accomplished through the Unity3D game engine system.

Enemy Controller
The enemy controller is a module that manages enemy game objects. In this projects enemies are any moving game object that is not the player, this includes zombies, Brainz, and Gainz.

Environment Controller
This controller manages environmental objects such as physics objects and stages. These objects do not have a Dynamic AI component, so are not included in the enemy controller.

GUI Controller
The GUI controller is a module that manages the on screen graphical elements. This includes updating HUD values during gameplay and managing menu options.

Player Controller
The player controller manages the player game object, the object that the user directly influences through inputs. This controller also includes the input controller, which manages user input, and player collision controller, which manages when the player collides with another object.

Light Controller
The Light Controller manages light objects, and runs the appropriate lighting shaders. The light controller makes sure to only spend processing time on lights that are within a player's view frame.

Major Data Structures

```
public class GameController : MonoBehaviour {
    private GameState gameState; // Game State datastructure
    private EnviroController envControl; // Updates environmental objects such as boxes.
    private GUIController hud; // Manages and updates game hud(heads up display).
    private LightController lights; // Manages lighting shaders
    private Spawner enemySpawner; // Spawns and manages enemy zombies
    private Spawner powerupSpawner; // Spawns and manages powerups
    private Spawner bngSpawner; // Spawns and manages brainz and gainz
    private GameObject enemies[]; // Array of enemies
    private GameObject brainz[]; // Array of Brains objects
    private GameObject gainz[]; // Array of Gainz objects
    private GameObject powerups[]; // Array of active powerups
    private GameObject player; // Player game object
    public void Start (); // Initializes class
    public void Update ();  // Update is called once per frame
}
```

```
// Tracks and manages Variables that are contained within the game.
public class GameState : MonoBehaviour{
    public bool game_over = false; // Set to true and game ends
    public int brainz = 0; // Brainz Counter
    public int gainz = 0; // Gainz Counter
    public void Start (); // Initializes class
    public void Update ();  // Update is called once per frame
}
// Enum type that contains valid powerups
public enum PowerUp{
    NONE,
    BLINK,
    INVISIBILITY,
    STUN,
    SECOND_WIND,
    INVULNERABLE
};
// Tracks state of player character
public class PlayerController : MonoBehaviour
            public PowerUp power_up = PowerUp.NONE;
            public float power_up_time_remaining = 0.0f;
            public float stamina_recovery_rate = 1.0f;  // how much stamina you recover per sec.
            public float max_stamina = 5.0f;  // Maximum amount of stored stamina.
            public float stamina; // How much stamina you currently have
            private bool sneaking = false;// Player sneaking state
            public bool running = false; // Player running state.
            private float cooldown  = 0.0f; // Cooldown in seconds till player can activate an ability
again.
            public void Start (); // Initializes class
            public void Update ();  // Update is called once per frame
            public void UpdateTimers(); // Updates all timers
            public void UpdatePowerupTimer(); // Updates powerup time remaining timer
            public void UpdateCooldowns(); // Updates Cooldown timer.
            public void UpdateStamina(); // Updates Stamina

            public void DeletePowerup(); // Removes current player powerup
            public void SetPowerUp(PowerUp set); // Sets current player powerup
            public void SetSneaking(); // Sets sneaking attribute and sets running attribute to false.
            public bool GetSneaking(); // Gets sneaking state.
```

```
        public void SetRunning(); // Sets running attribute to true and sets sneaking attribute to
        false.
        public bool GetRunning(); // Get Running state
        public void SetWalking(); // Sets walking state which is running and sneaking attributes
        false.
        public bool GetWalking(); // Gets walking state, returns true iff running and sneaking
        false.
}

 // Module that spawns and manages game objects
public class Spawner : MonoBehaviour
    public int max_count = 100; // Max number of things to spawn.
    public float spawn_frequency = 1; // How often we will spawn
    public GameObject[] objects; // Datastructure of all spawned objects
    public float min_distance = 0; // How close can we spawn
    public float max_distance = 0; // How far away can we spawn
    public GameObject spawn_point; // Around What we are Spawning
    private int counter = 0;    // How many objects have spawned so far
    private float timer = 0.0f;  // How long since last spawn.
    void Start ();  // Initializes Class
    void Update ();  // Update is called once per frame.  Checks if object ready to spawn
    void Spawn(Vector3 position, GameObject obj); // Spawns Object, calle
    Vector3 GetValidRandomEnemySpawnPoint();  // Returns a random point to spawn based
on min and max distance.
}
// Module that manages enemy movement and AI.  This coupled with EnemyMovement form
the EnemyController module.
public class EnemyAI : MonoBehaviour {
    public float aggro_range = 5.0f; // Range within which enemy will ‘aggro.’
    public float max_speed = 2.0f; // Maximum speed in feet per second object can travel
    public float acceleration = 1.0f; // maximum acceleration in feet per second squared object
can travel.
    private Vector3 velocity = new Vector3(0,0,0);  // Current velocity vector for this frame.
    private GameObject player; // Reference to player game object
    public void Start (); // Initializes class
    public void Update ();    // Update is called once per frame
    private bool IsAggroed(Vector3 player_pos, Vector3 enemy_pos); // Calculates distance
between player and enemy, and returns aggroed boolean.
}
```

```
// Module that manages Brainz/Gainz movement and AI.
public class BnGControl : MonoBehaviour {
    public float flee_range = 5.0f; // Range at which object begins to run away
    public float max_speed = 2.0f; // Max speed at which object can move in feet per second.
    public float acceleration = 1.0f; // Max acceleration object can move in feet per second
squared.
    public Vector3 velocity = new Vector3(0,0,0); // Current velocity vector of object for this
frame.
    private GameObject player; // Reference to player object.
    public bool fleeing = false; // Fleeing status of object, set to true once player within flee range.
    public void Start ();     // Initializes class.
    public void Update ()   // Update is called once per frame
}

// Oscillates and spins Game Object around its start point
public class Oscillate : MonoBehaviour {
    public float oscillation_length = 1.0f;  // How far an object oscillates in feet.
    public float spin_frequency = 1.0f;  // How long it takes object to spin 360 degrees in seconds
    private float start_y;  // Starting location of object

    // Initializes object position
    public void Start ();

    // Update is called once per frame and updates object position based on current time.
    public void Update () ;
}

public class PlayerCollision : MonoBehaviour {
    private PlayerState playerState; // Use this for initialization
    public void Start (); // Initializes player collision and grabs player set.
    public void Update (); // Called on update, currently does nothing
    private void OnControllerColliderHit(ControllerColliderHit collision); // Event callback when
object collides with player
}
```

```
public class PlayerMovement : MonoBehaviour
{
    public AudioClip shoutingClip;      // Audio clip of the player shouting.
    public float turnSmoothing = 15f;   // A smoothing value for turning the player.
    public float speedDampTime = 0.1f;  // The damping for the speed parameter
    public float base_speed = 5.0f; // Base/Unmodified
    private PlayerState playerState; // Pointer to player state object
    private Animator anim;              // Reference to the animator component.
    private HashIDs hash;               // Reference to the HashIDs
    private InputManager inputManager;  // Manages and stores inputs.
    public void Start();  // Initializes class.
    public void Update (); // Called every frame, manages audio, animation, and player position
    private void Awake(); // Manages initial animation setup.
    private void FixedUpdate(); // Updates player position and updates animation state.
    private void AudioManagement (bool shout); // Manages player noise, like shouting and
footsteps.
}

 // Manages controller inputs.
public class InputController : MonoBehaviour {
    private GameObject player;
    private PlayerState playerState;
    void Start () ; //  Initializes class.
    void Update ();   // Update is called once per frame. Checks each key and updates method
}
```

Method Description

Class - All Classes.
Method: Start ();
Visibility: Public.
Return Type: None.
Parameters: None
Description - Initializes the class instance, called by game engine. This is a built in method handled by the Unity game engine.

Class - All Classes.
Method: Update ();
Visibility: Public.
Return Type: None.
Parameters: None
Description - Called once per frame and updates Class instance. This is a built in method handled by the Unity game engine.

Class - PlayerController.
Method: UpdateTimers ();
Visibility: Public.
Return Type: None.
Parameters: None
Description - Updates all timers, anything that is waiting for a specific time to trigger.

Class - PlayerController.
Method: UpdateCooldowns ();
Visibility: Public.
Return Type: None.
Parameters: None
Description - Updates the timers for ability cooldowns.

Class - PlayerController.
Method: UpdatePowerupTimer ();
Visibility: Public.
Return Type: None.
Parameters: None
Description - Updates the timers for powerup cooldowns.

Class - PlayerController.
Method: UpdateCooldowns();
Visibility: Public.
Return Type: None.
Parameters: None
Description - Updates the timers for cooldowns.

Class - PlayerController.
Method: UpdateStamina ();
Visibility: Public.
Return Type: None.
Parameters: None
Description - Updates stamina. If player is running, time is reduced from stamina, if player is not running, time is added to stamina.

Class - PlayerController.
Method: DeletePowerup ();
Visibility: Public.
Return Type: None.
Parameters: None
Description - Removes a powerup from PlayerController, and sets powerup cooldown to zero.

Class - PlayerController.
Method: SetPowerUp (PowerUp set);
Visibility: Public.
Return Type: None.
Parameters: A powerup to be set of type PowerUp
Description - Sets the current powerup, and sets powerup cooldown to its maximum.

Class - PlayerController.
Method: SetSneaking ();
Visibility: Public.
Return Type: None.
Parameters: None
Description - Sets sneaking attribute to true and sets running attribute to false.

Class - PlayerController.
Method: SetRunning ();
Visibility: Public.
Return Type: None.
Parameters: None
Description - Sets running attribute to true and sets sneaking attribute to false.

Class - PlayerController.
Method: SetWalking();
Visibility: Public.
Return Type: None.
Parameters: None
Description - Sets sneaking attribute to false and sets running attribute to false.

Class - PlayerController.
Method: GetSneaking ();
Visibility: Public.
Return Type: Bool.
Parameters: None
Description - Gets sneaking attribute.

Class - PlayerController.
Method: GetRunning ();
Visibility: Public.
Return Type: Bool
Parameters: None
Description - Gets running attribute.

Class - PlayerController.
Method: GetWalking();
Visibility: Public.
Return Type: Bool.
Parameters: None
Description - Gets walking state, which is set to true if running and walking are both set to false.

Class - Spawner.
Method: Spawn();
Visibility: Public.
Return Type: Void.
Parameters: Vector position of object, and game object to spawn.
Description - Spawns GameObject object at position.

Class - Spawner.
Method: GetValidRandomEnemySpawnPoint();
Visibility: Public.
Return Type: Vector3.
Parameters: None.
Description - Returns a random spawn point generated from class parameters initialized in start();

Class - PlayerController.
Method: OnControllerColliderHit();
Visibility: Public.
Return Type: None.
Parameters: Collision hit.
Description - Event callback. Manages collision with player game object.

Detail Design
This section contains diagrams and pseudo-code depicting low-level details of the BrainZ & GainZ system.

Below is the pseudo-code for the PlayerCollsion, EnemyAI, and PlayerMovement.

```
PlayerCollision()
{
        if object.deadly is true
        {
                if powerup is invulnerable then {
                        Destroy object
                else
                {
                        player.dead is true
                }
        }
        if object is brainz {
                brainz++
        }
        if object is gainz {
                gainz++
        }
        if object is powerup {
                if powerup is blink then {
                        player.powerup is blink
                }
                if powerup is invulnerable then {
                        player.powerup is invulerable
                }
                if powerup is invisibility then {
                        player.powerup is invisibility
        }


}
```

```
EnemyAI()
{
    if player.powerup is true {
        if powerup is blink then {
            enemy.destroy is true
        }
        if powerup is invulnerable then {
            object.deadly is false
        }
        if powerup is invisibility then {
            enemy.follow is false
        }
    }
    else {
        enemy.follow is true
        if player.collide is enemy then {
            player.dead is true
        }
    }
}

PlayerMovement()
{
    if button.input is “NONE” then {
        player.movement is running
        if button.input is “SPACE” then {
            player.jump is true
        }
    }
    if button.input is “CTRL” then {
        player.movement is walk
        if button.input is “SPACE” then {
            player.jump is true
        }
    }
    while player.movement is running or walk {
        mouse.look is player.position
    }
}
```

Below are some activity charts for BrainZ & GainZ.

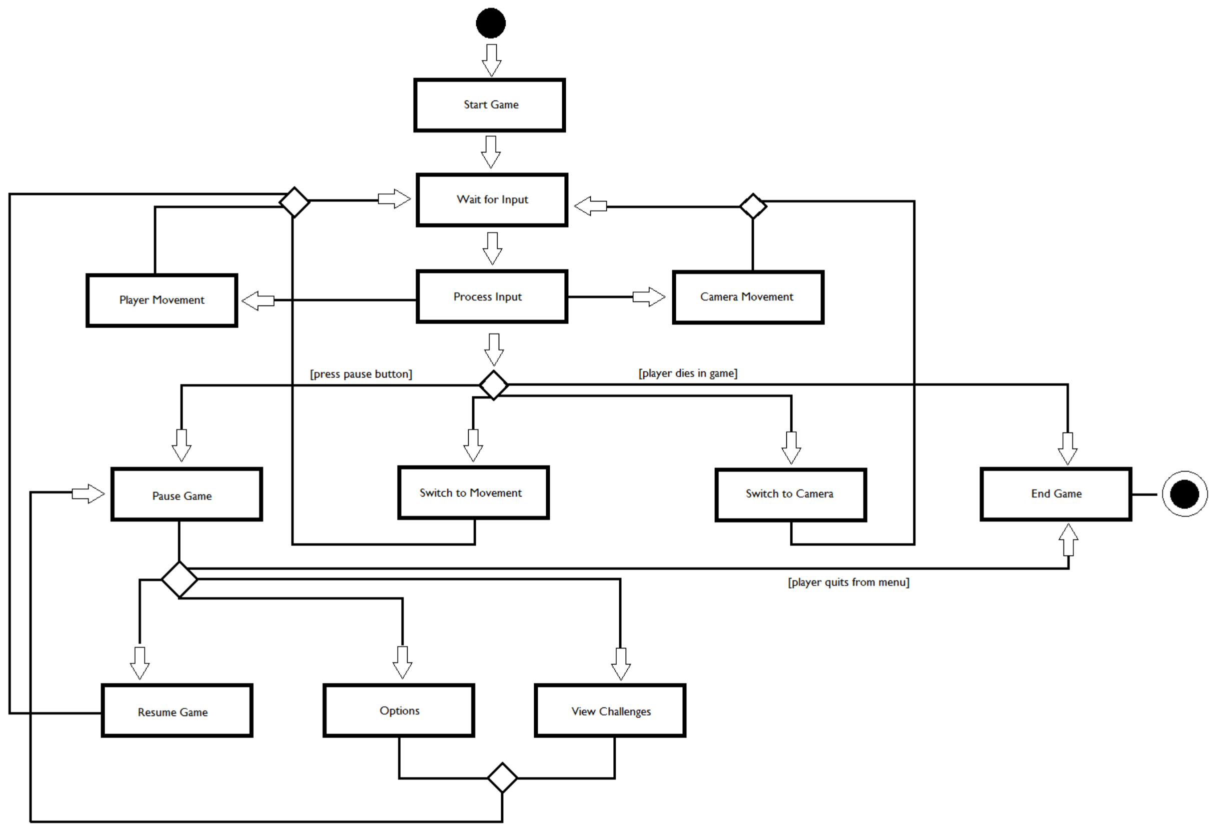


Figure 2: BrainZ & GainZ activity chart during Gameplay

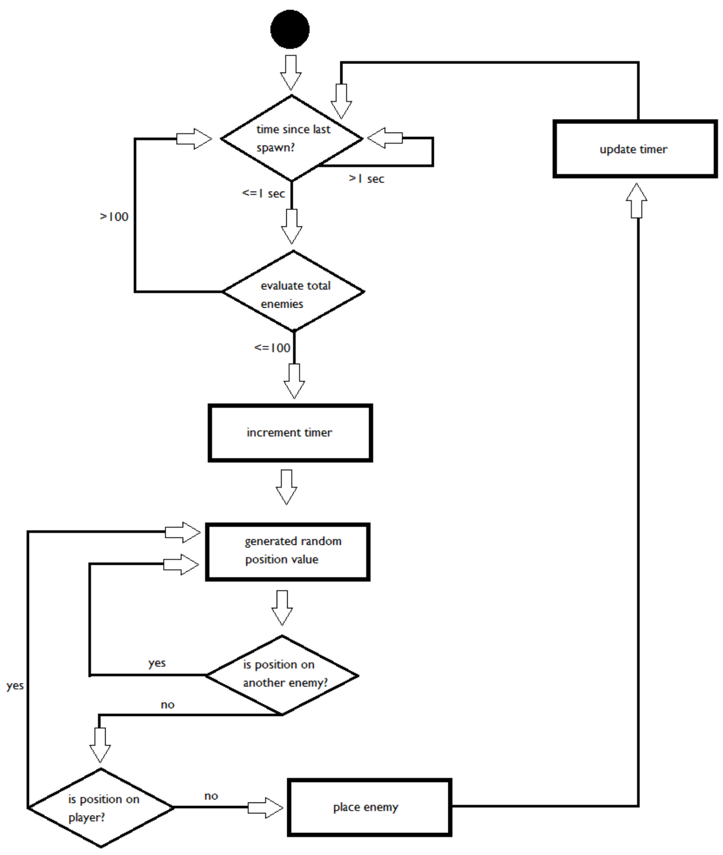


Figure 3: BrainZ & GainZ activity chart for spawning enemies

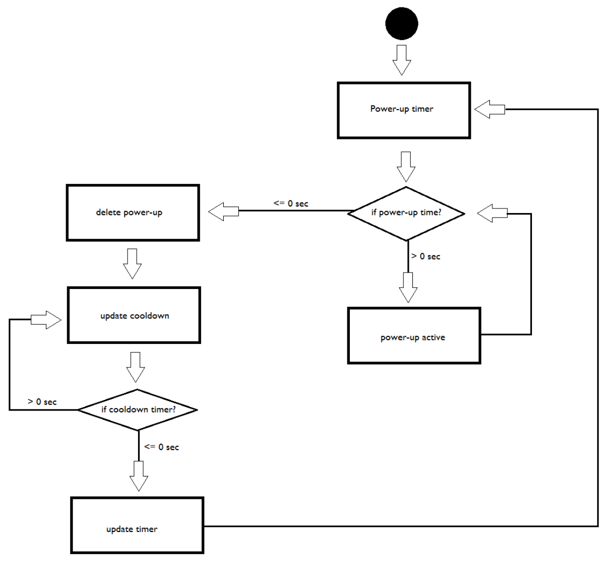


Figure 4: BrainZ & GainZ activity chart for power-ups

User Interface Design

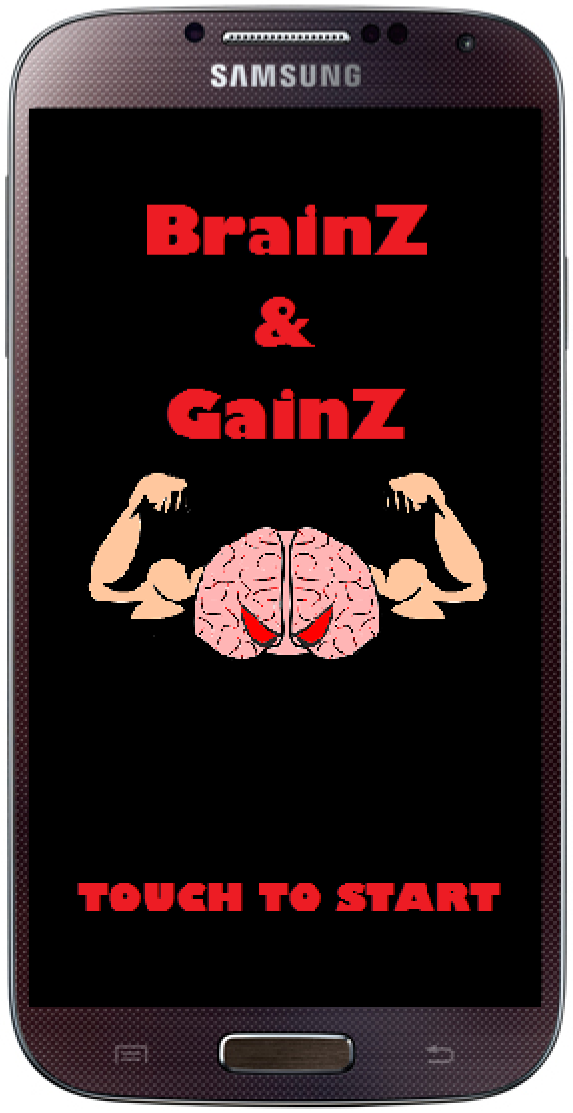


Figure 5: Snapshot of the start screen of BrainZ & GainZ

Figure 5 displays the start screen of BrainZ & GainZ. Any data that must be loaded or obtained from the server will be done after the player touches the screen before they are brought to the main menu.

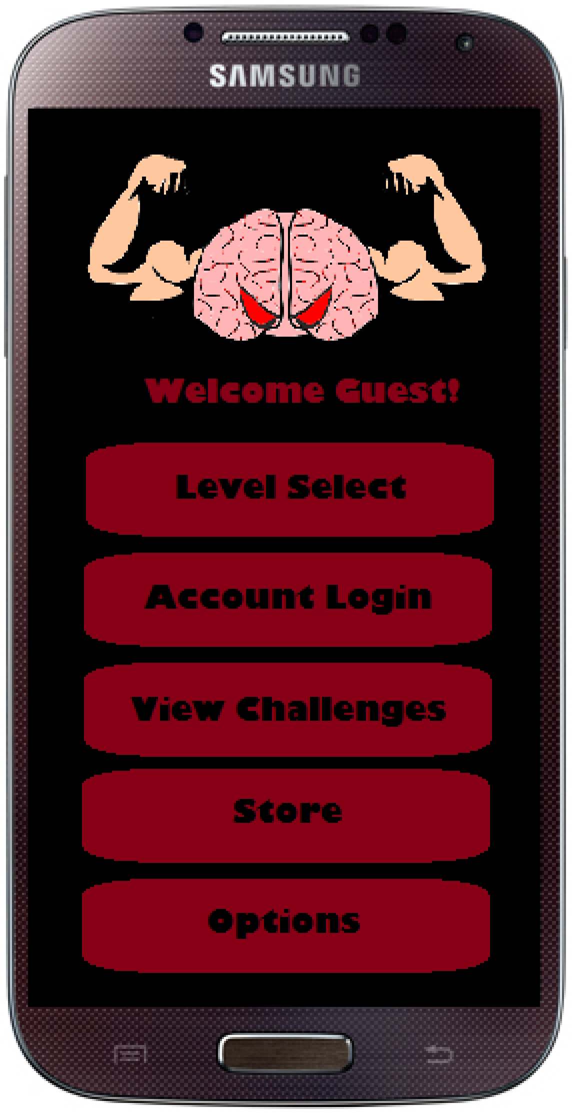


Figure 6: Snapshot of the main menu of BrainZ & GainZ

Figure 6 displays the main menu of BrainZ & GainZ. The player can tap the "level select", "account login", "view challenges", "store", and "options" button to take them to the corresponding screen. By default the game will display "Welcome Guest!" until the user logs in. After it would display "Welcome <USERNAME>!".

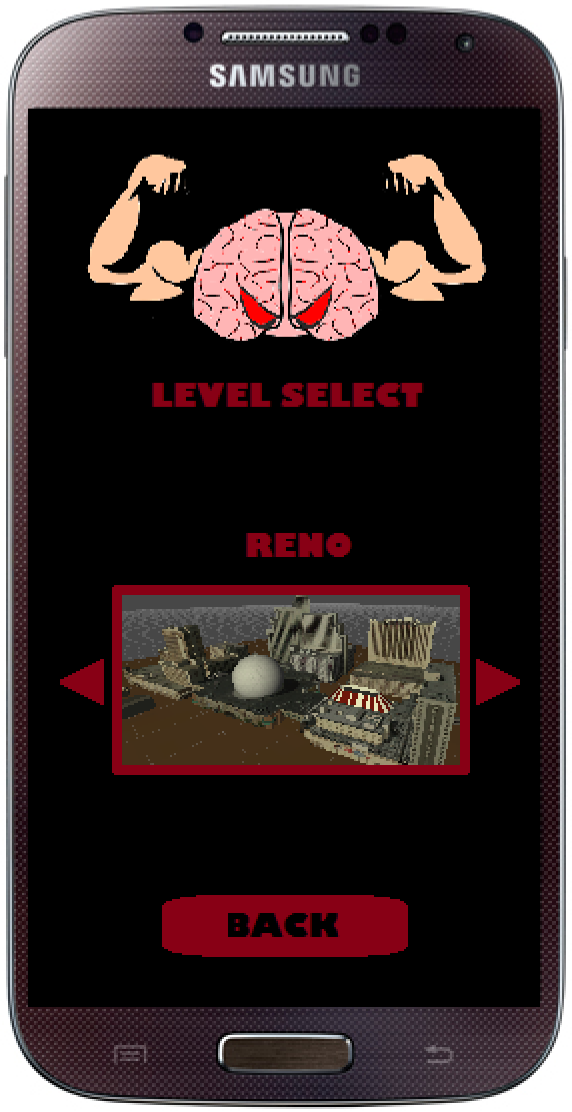
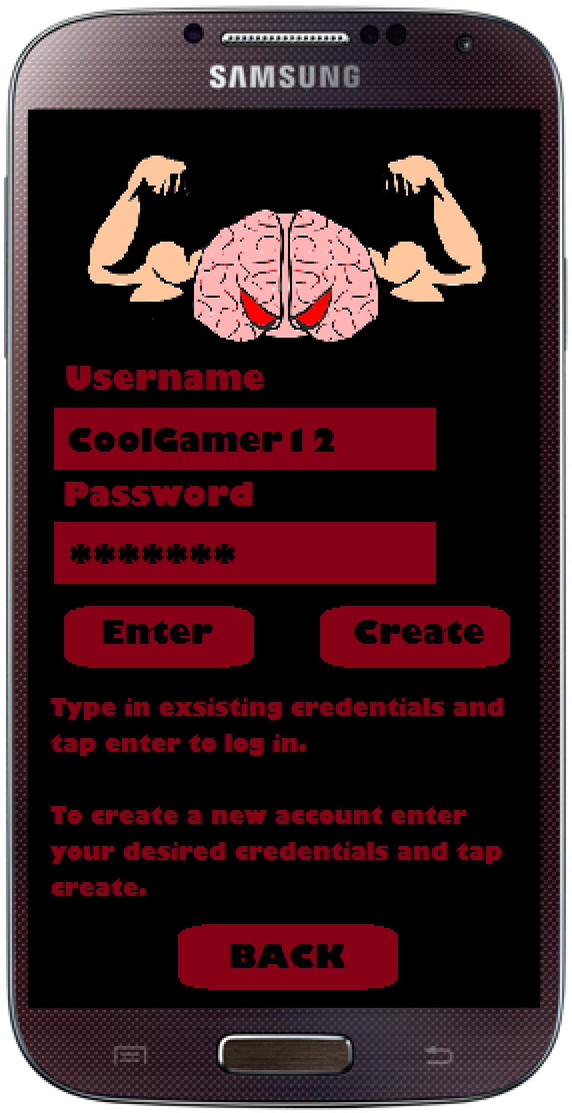


Figure 7: Snapshot of the level select screen of BrainZ & GainZ

Figure 7 displays the level select screen of BrainZ & GainZ. The player can play the currently displayed level by tapping on the image of the level or can cycle through all the levels by tapping either arrow. Initially if a level has not been unlocked it will show a question mark and not display the name but under it will display the requirements to unlock the level. If downloadable content is to be completed before the end of the semester there will be an extra button to take the player to the store to purchase more levels.

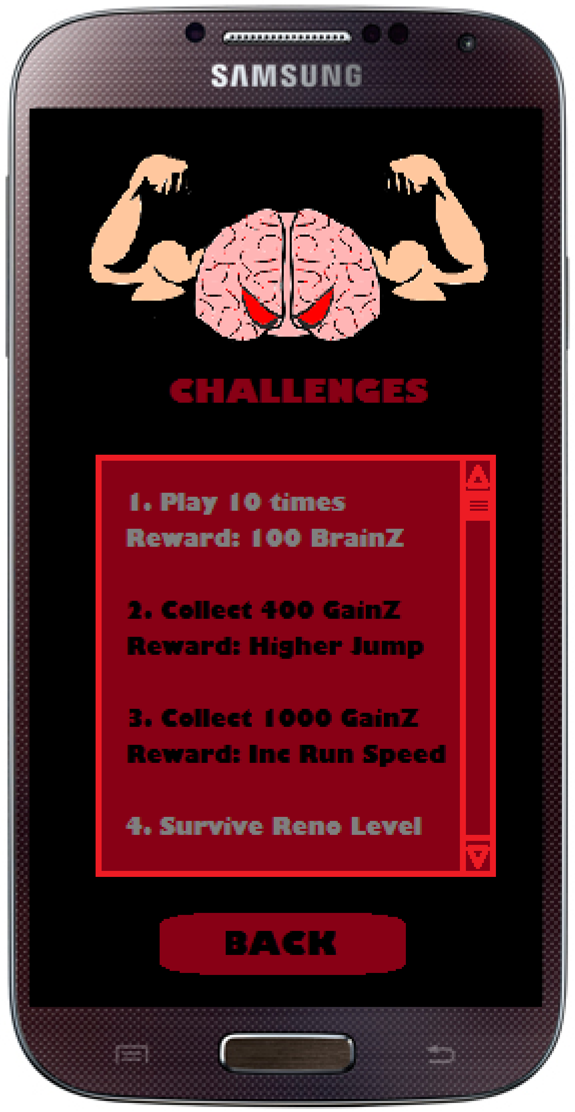


Type in exsisting credentials and tap enter to log in.

To create a new account enter your desired credentials and tap create.

Figure 8: Snapshot of the login screen of BrainZ & GainZ

Figure 8 displays the login screen of BrainZ & GainZ. The player can either create a new account or login to an existing account. If the player wishes to login to an existing account the will enter their credentials and tap “enter”. If the credentials match an entry in the BrainZ & GainZ database then the user will be logged in, if not it will prompt the user that the credentials they entered were not valid. If the player wishes to create a new login the player will enter in their desired credentials and tap “create”. If there is already an existing user with the same the desired username then the player will be prompted to pick a new username.



1. Play 10 times
Reward: 100 BrainZ

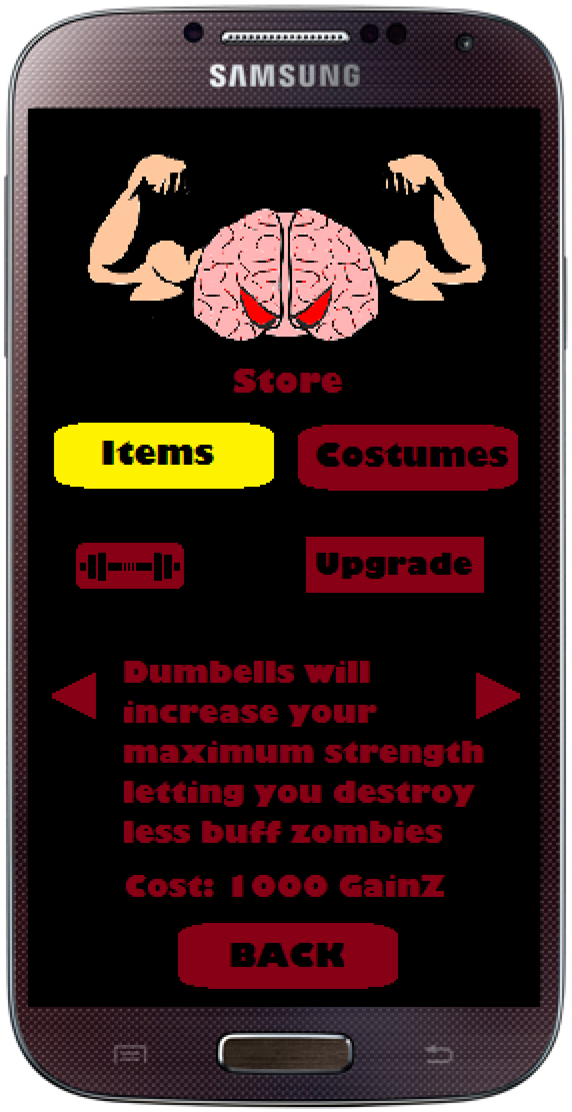
2. Collect 400 GainZ
Reward: Higher Jump

3. Collect 1000 GainZ
Reward: Inc Run Speed

4. Survive Reno Level

Figure 9: Snapshot of the challenges screen of BrainZ & GainZ

Figure 9 displays the challenges screen of BrainZ & GainZ. The player will be able to view all challenges of the game and their corresponding reward for completion. Some rewards may be unlocking new levels, costumes, items, BrainZ, or GainZ. Challenges that have been completed will show up in gray and challenges to be completed will show up in black.



Dumbells will increase your maximum strength letting you destroy less buff zombies

Cost: 1000 GainZ

Figure 10: Snapshot of the store screen of BrainZ & GainZ

Figure 10 displays the store screen of BrainZ & GainZ. The user will be able to purchase items and costumes that may add cosmetic effects or stat increases to their character. When a player tries to purchase an item the game will check to see if their account has the required amount of BrainZ or GainZ needed to unlock the desired item or costume.

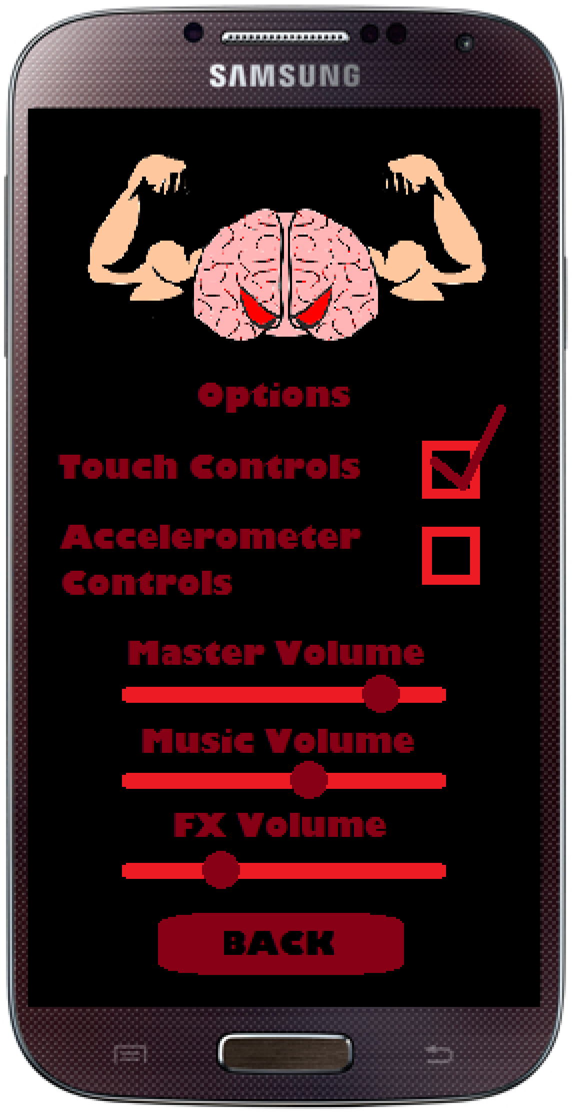


Figure 11: Snapshot of the options screen of BrainZ & GainZ

Figure 11 displays the options screen of BrainZ & GainZ. In the options menu the user can select which control input they wish to use. The player can also adjust various volume settings as well.

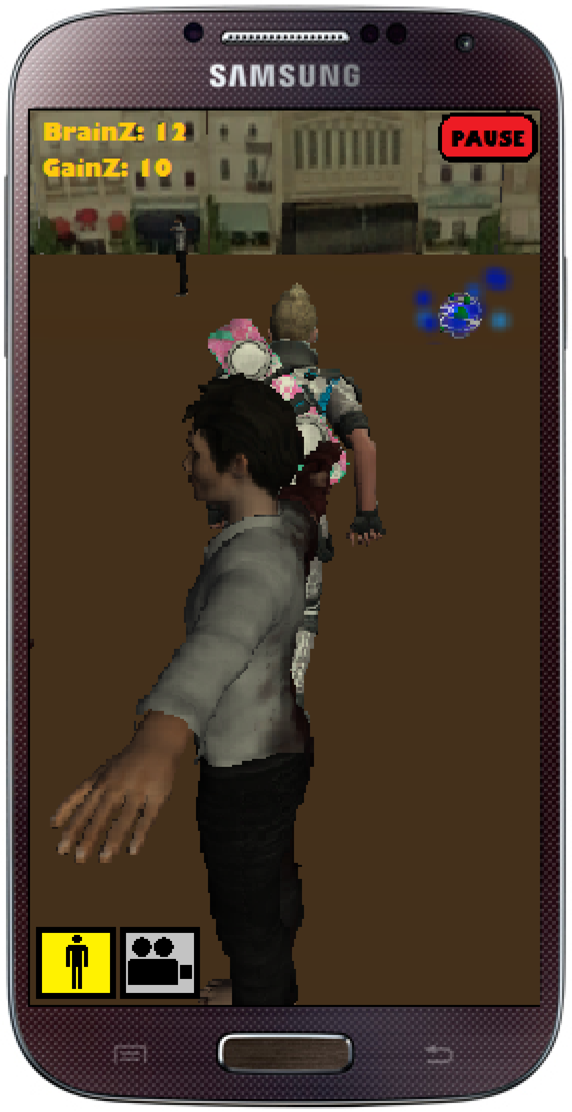


Figure 12: Snapshot of the gameplay screen of BrainZ & GainZ

Figure 12 displays a sample of the gameplay screen of BrainZ & GainZ. The player can control the game by tapping anywhere on the screen and dragging their finger in the direction they would like to turn. By tapping and holding onto the screen the character will continue to run. The player can tap their desired control of the player or camera at the bottom left of the screen and the current control is colored yellow. The player can also touch the pause button to pull up the gameplay pause menu options detailed in figure 13.

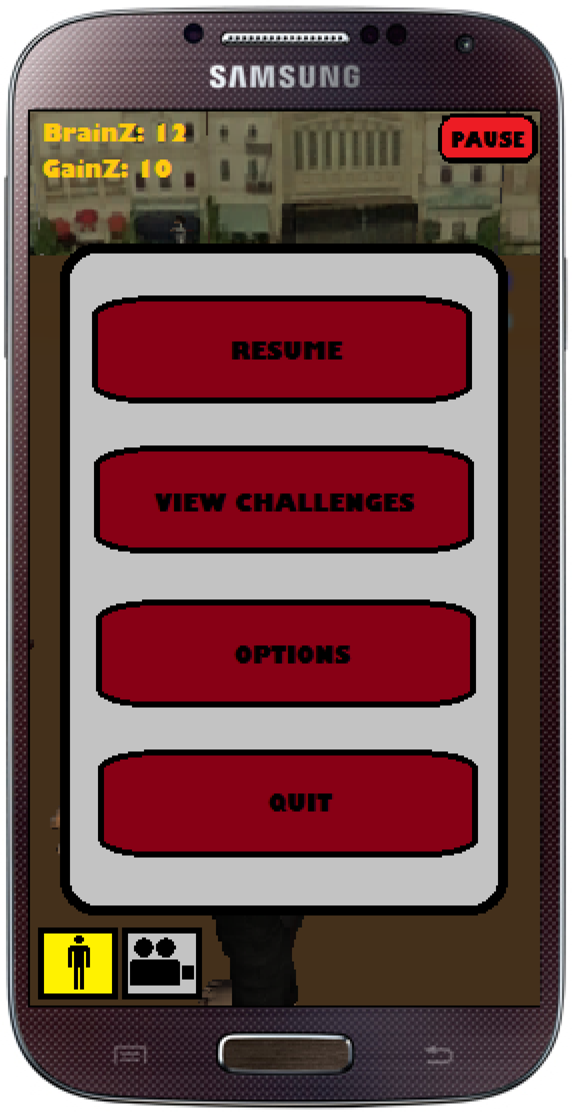
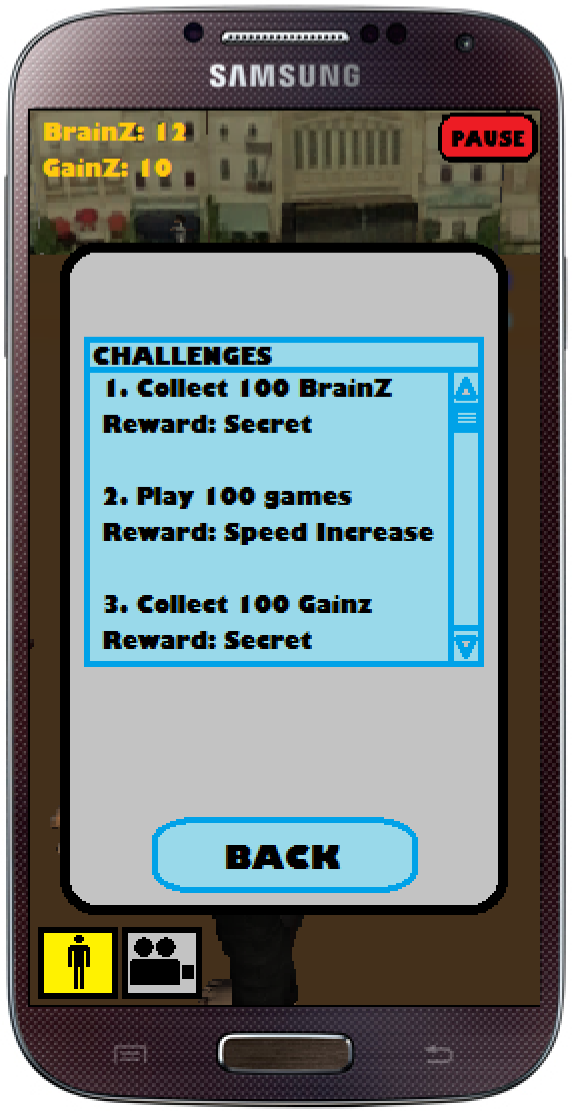


Figure 13: Screenshot of the pause menu screen during gameplay of BrainZ & GainZ

Figure 13 displays the gameplay pause menu of BrainZ & GainZ. The player can resume their game by tapping "resume", display the challenges by tapping "view challenges", change game options by tapping "options", or be brought back to the main menu by tapping "quit".



CHALLENGES

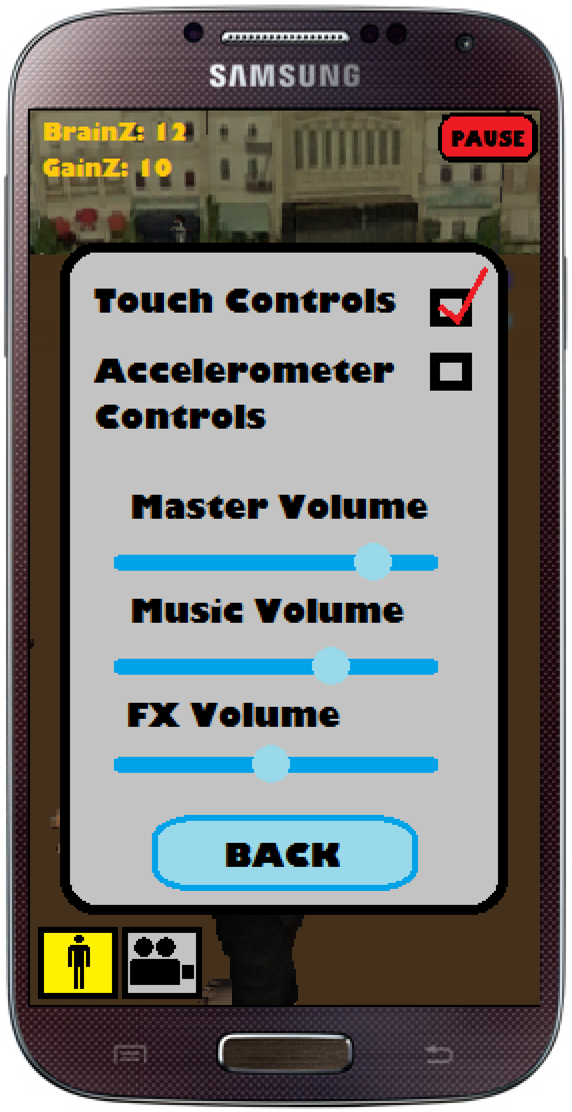
1. Collect 100 BrainZ
Reward: Secret

2. Play 100 games
Reward: Speed Increase

3. Collect 100 Gainz
Reward: Secret

Figure 14: Snapshot of the challenges menu during gameplay of BrainZ & GainZ

Figure 14 displays the challenges menu during gameplay of BrainZ & GainZ. The player can view challenges to complete during gameplay to determine which challenges they can complete during their current game.


BrainZ: 12
GainZ: 10

Touch Controls ✓

Accelerometer Controls ☐

Master Volume

Music Volume

FX Volume

Figure 15: Snapshot of the options menu during gameplay of BrainZ & GainZ

Figure 15 displays the options menu during gameplay of BrainZ & GainZ. The player can change volume controls or their current game input controls.

Glossary

Accelerometer - A built-in electronic component in mobile devices that measures tilt and motion. It is also able to detect rotation and motion gestures such as swinging or shaking.

AI - Artificial intelligence is the intelligence exhibited by machines or software.

Alpha testing - Testing or actual operation by potential end users. Alpha testing is often employed for off the shelf software as a form of acceptance testing before being released to a wider audience for testing.

Android OS - A Linux-based operating system developed by Android, Inc. primarily used for touchscreen mobile devices.

Animation - A sequence of successive states or positions of a game object to create the illusion of movement.

C# - A syntax and semantics built off of C++ and commonly used in coding Unity game objects.

Controller - A subsystem that manages all aspects pertaining to one area of emphasis.

First person - A graphical perspective from the viewpoint of the player.

Game object - A base class for all entities in Unity scenes.

GUI - A graphical user interface which allows users to interact with devices or software.

IDE - An integrated development environment software that usually consists of source code editors and build automation tools. IDE's are used by computer programmers for software development.

iOS - Apple's mobile operating system used to run Apple products such as the iPhone and iPad.

Porting - Adapting software to be used by different computing environments it was originally designed for.

Prefab - a reusable game object in Unity that can be inserted into any number of scenes as many times as desired.

Third person - A graphical perspective from a fixed distance away from the player.

Scene - The game space in Unity in which objects can be placed. Scenes are commonly used in Unity to implement individual levels or menus.

Texture - The feel, appearance, or consistency given to a surface or game object.

Unity - A cross-platform game engine developed by Unity Technologies with built-in IDE for video games and simulations.

New Glossary additions

Beta Testing - The last stage of testing, and normally can involve sending the product to test sites outside the company for real world exposure or offer the product as a free trial to download.

Intuitive - easy to use and understand.

Module - any of a number of distinct but interrelated units from which a program may be built up or into which a complex activity may be analyzed.

Oscillate - vary in magnitude or position in a regular manner around a central point.

Power-up - an object that can instantly benefit or add extra abilities to the game character as a game mechanic.

Pseudo-code - an informal high-level description of a computer program or algorithm.

Contribution of team members

Albert - Major data structures used in your project, The structuring of your program units

Brian - Abstract, high-level diagram, User interface

Collin - Introduction, Detailed Design, Glossary Updates, Contributions of team members

Resources

Despain, Wendy. *100 Principles of Game Design*. New Jersey: New Riders, 2013. eBook.