# Test Plan
# Group 3

Title: Select Pinball
Actors: User
Requirement: Endless Pinball
Main Scenario:

1. User selects the pinball button
2. System displays all pinball color options
3. User selects desired pinball
4. System displays confirmation button a)User selects confirm b)or cancel
5. a)System updates pinball gameObject with new pinball or b) System keeps pinball as the same

Alternatives:
2a. System is unable to retrieve any pinball color options
2b. System displays error message to user
2c. System sends the user back to the main menu
3a. User selects pinball that is already set as desired pinball
3b. System sends message informing user
3c. System sends the user back to the pinball menu

Test situations:
User successfully selects desired pinball color
System is unable to retrieve pinball color options
User selects pinball that was already selected

Test Coverage:
Main: 5
Alternatives: 6
Test coverage covers all 11 cases
100% coverage of cases

Title: Selecting Board Theme
Actors: User
Requirement: Endless Pinball
Main Scenario:
1.User selects change board option
2.System displays all board options.
3.User selects desired board
4.System displays confirmation button
5.User confirms changes.

Alternatives:
2a. System is unable to display board theme options
2a2. System displays error message
2a3. System sends user back to main menu
3b.User selects current board theme as new board theme
3b2. System sends message informing user
3b3.System sends user back to main menu

Test situations:
User successfully  selects desired board theme
System fails to retrieve board theme options
User selects board theme that was already being used

Test Coverage:
Main: 5
Alternatives: 6
Test coverage covers all 11 cases
100% coverage of cases

Title: Log In to Profile
Actors: User
Requirement: Endless Pinball
Main Scenario:

1.) User selects settings menu
2.) System displays many various options
3.) User selects profile menu
4.) System accepts request
5.) User types in profile information
6.) Systems connects to profile

Alternatives:
2a. System is unable to display settings menu
2a2. System displays error message
3b.User inputs incorrect profile information
3b2. System displays error message

Test situations:
User successfully logs into Profile
System fails to display profile menu
User inputs incorrect profile information

Test Coverage:
Main: 6
Alternatives: 4
Test coverage covers all 10 cases
100% coverage of cases

Title: Selecting a Song
Actors: User
Requirement: Endless Pinball
Main Scenario:

1. User selects the music change button
2. System displays all Endless Pinball tracks
3. User selects desired track
4. System displays confirmation button and cancel button
5. a)User selects the confirmation button or b) user selects cancel button
6. a)System plays audio of newly selected music or b) System resumes currently playing audio

Alternatives:

2a. System is unable to retrieve any song options
2b. System displays error message to user
2c. System sends the user back to the main menu
3a. User selects song that is already set as desired song to play
3b. System sends message informing user
3c. System sends the user back to the song selecting menu

Test situations:
User successfully selects desired song
System is unable to retrieve song options
User selects song that was already selected previously

Test Coverage:
Main: 5
Alternatives: 6
Test coverage covers all 11 cases
100% coverage of cases

Title: Uploading a High Score
Actors: User
Requirement: Endless Pinball
Main scenario:

1. User selects the “Upload” button
2. System verifies if the user is logged in
3. User enters a name to be placed on the leaderboard

4. System uploads score to the website and displays “Save Name” button and “Home” button
5. User selects “Save Name” button

Alternatives:
2a. System is unable to verify the user’s login information
2a2. System displays error message and returns user to previous menu
3a. User enters an invalid name
3a2. System displays invalid name message and the user must try again
5a. User selects “Home”
5a2. System sends user back to main menu

Test situations:
User successfully uploads a high score
System is unable to upload score
User chooses invalid name

Test Coverage:
Main: 5
Alternatives: 6
Test coverage covers all 11 cases
100% coverage of cases

## Unit Testing

Test #1 Remove Score
This test checks if the user has a score associated with their account on the leaderboard. Then, it checks to see if it can delete it or not. If the account does not have a score, it then sends an error message to the system.

```
#[test]
fn remove_score() {
    let d = NaiveDate::from_ymd(2015, 6, 3);
    let t = NaiveTime::from_hms_milli(12, 34, 56, 789);

    let dt = NaiveDateTime::new(d, t);

    let a = User::add(2, format!("UserA").to_string());
    let b = Score::add_score(a,dt);
    assert_eq!(Score::remove_score(b),2);
}
```

Test # 2: Insert User
This Test checks if the user's id is equal to 2. If not, it does not allow the user to be entered into the website's database.

```
#[test]
fn insert_user() {
    let a = User::add(2, format!("UserA").to_string());
    assert_eq!(a.get_id(),2);
}
```

Test #3: Get name
This test checks the user's name is the same as the one on the website. If the name gathered by the code is not the same, it sends a message to the system.

```
#[test]
fn get_name(){
    let a = User::add(2, format!("UserA").to_string());
    assert_eq!(a.get_name(),"UserA");
}
```

Test #4: World Derender
This test checks if the world de-renders correctly, and checks if the application properly updates the game state.

```
#[test]
fn world_derender(){
    let mut world = World::default();
    let mut update_stage = SystemStage::parallel();

    update_stage.add_system(setup);

    let mut game_id = world.spawn().insert(OnGameScreen).id();

    game_id = world.spawn().insert(OnPauseMenuScreen).id();

    update_stage.run(&mut world);

    assert!(world.get::<Component>(game_id).is_none());

}
```

Test #5: Test Play
This test checks whether the system correctly initializes a level in the game. If the level does not exist or fails to initialize, it sends a message to the system

```rust
#[test]
fn test_play(){
    let mut world = World::default();
    let mut update_stage = SystemStage::parallel();

    update_stage.add_system(setup);

    let click_id = world.spawn().insert(MenuButtonAction::Play).id();

    update_stage.run(&mut world);

    assert!(world.get::<Component>(click_id).is_some());

}
```

Test #6 Sound Stop

This test checks to see if the volume of the system is set to 0. If the check is returned False, it flags this to the system.

```rust
#[test]
fn sound_stop(){
    let mut world = World::default();
    let mut update_stage = SystemStage::parallel();

    update_stage.add_system(setup);

    let sound = world.spawn().insert(Volume(4)).id();

    update_stage.run(&mut world);

    world.insert_resource(stop_music);

    update_stage.run(&mut world);


    assert!(world.query::<&Volume>().iter(&world).is_paused(),True)


}
```

Test #7 World Render

This test checks to see if the components used in the world (game) are actually active or not. If not, it sends a message to the system.

```rust
#[test]
fn world_render(){
    let mut world = World::default();
    let mut update_stage = SystemStage::parallel();

    update_stage.add_system(setup);

    let game_id = world.spawn().insert(OnGameScreen).id();

    update_stage.run(&mut world);

    assert!(world.get::<Component>(game_id).is_some());



}
```

Test #8: Sound Decrease
This test checks to see if the system can properly decrease the volume of the application. If it cannot, it sends an error message to the system.

```
fn sound_decrease(){
    let mut world = World::default();
    let mut update_stage = SystemStage::parallel();


    update_stage.add_system(setup);

    let sound = world.spawn().insert(Volume(4)).id();

    let mut input = Input::<KeyCode>::default();
    input.press(VolumeDown);
    world.insert_resource(input);

    update_stage.run(&mut world);

    assert_eq!(world.get::<Volume>(sound).unwrap().0, 3)

}
```

Test #9: Delete user
This test checks if a user of a certain name exists, and if it does, then it checks if the user can be removed from the system. If it fails, it sends an error message to the system

```
    #[test]
    fn delete_user() {
        let a = User::add(2, format!("UserA").to_string());
        assert_eq!(a.remove_user(),2);
    }
```

Test #10:  Insert Score
This test shown inserts the score with a user into the database. This is used to test the database's capabilities

```rust
#[test]
fn insert_score() {
    let d = NaiveDate::from_ymd(2015, 6, 3);
    let t = NaiveTime::from_hms_milli(12, 34, 56, 789);

    let dt = NaiveDateTime::new(d, t);

    let a = User::add(2, format!("UserA").to_string());
    assert_eq!(Score::add_score(a,dt),true);
}
```

Test #11: Insert Image

This test checks if the system is able to insert an image into the system.

```rust
#[test]
fn insert_image() {
    assert_eq!(2 + 2, 4);
}
```