<^> TRIFORCE <^>

Analysis Workflow

by Gabe Pike

CMPS 335

Spring 2012

Team 10

Other Team Member: Brandon Hinesley

Section 2: Analysis Workflow

2.1 Analysis Workflow

2.1.1 Functional Modeling

Normal scenarios

1. Game starts and shows the main menu
2. Player selects the Play button
    - 2.1. Game is loaded and customized based on selected settings
    - 2.2. Timer counts down from 3 then starts the game
    - 2.3. Player moves with cursor
    - 2.4. Player swaps blocks
    - 2.5. Execute the combo detection subroutine
    - 2.6. If a combo is detected:
        - a. Grid changes state to COMBO
        - b. Blocks in the combo are changed to state COMBO
        - c. After S seconds the blocks break
        - d. Execute the fall detection subroutine (goto 2.7a)
    - 2.7. If no combo is detected:
        - a. Execute the fall detection subroutine
        - b. If a fall is detected:
            1. Set the blocks that need to fall to the FALL state
            2. Make the blocks fall by Y pixels every S milliseconds until they land on a non-disabled block.
            3. Execute the combo detection subroutine on each FALL block
            4. Increment the chain counter for each combo detected and goto 2.6
            5. Set the chain counter to 0 if no combo detected
            6. Set Grid's state to PLAY
    - 2.8. Player selects the Pause button
        - a. The game's state changes to PAUSE
        - b. The blocks are not displayed to prevent cheating
        - c. The player selects the Pause button again
        - d. The game's state changes to PLAY
        - e. Normal gameplay resumes

2.9. Player selects the Menu button
    a. Game objects are deleted
    b. The player is returned to the main menu
2.10. Player selects the Quit button
    a. The game immediately exits

3. Player selects the Settings button
    3.1. The settings menu is loaded
    3.2. Player toggles sound on/off
    3.3. Player toggles mouse on/off
    3.4. Player changes key bindings
    3.5. Player clicks the Save button
        a. Settings are saved
        b. Main menu is loaded
4. Player selects the Help button
    4.1. Some text and graphics are displayed that explain how to play the game
    4.2. Player selects the Back button
    4.3. Main menu is loaded
5. Player selects the About button
    5.1. A short description of the game, credits, and licensing information is displayed on the screen.
    5.2. Player selects the Back button
    5.3. Main menu is loaded
6. Player selects the Exit button
    6.1. Game immediately exits.

Exception scenarios

Early Quit:

1. Player starts playing the game
2. Player clicks Quit
3. Game immediately exits

Player intentionally loses

1. Player holds the push-row button
2. The blocks hit the top very quickly
3. Player loses
4. Game over screen is displayed
5. Player returns to main menu

Player gets very large chains

1. The player is skilled enough to chain many combos together
2. The game is able to handle it because combos are detected after every combo occurs.

2.1.2 Entity Class Modeling

Noun extraction

When the **game** is loaded, the **configuration file** is read and saved into **settings**. Then the **player** is presented with the **main menu**. The **player** can click **buttons** to take them to other **menus** or to play the **game**. The **settings menu** lets the user configure **options**. **Options** are saved to a **configuration file**. The **help menu** displays explains how to play the **game**. The **About menu** displays gives some background information about the **game**. The **player** clicks the **play button** and to start the **game**.

The **game play area** consists of the of a **grid** of **blocks**. The **player** controls a **cursor** and can use it to swap **blocks**. If at least 3 **blocks** are lined up, a **combo event** occurs. If there are disabled **blocks** under an enabled **block**, then a **fall event** occurs. A **fall event** can only occur after the player swaps or a **combo event.** A **combo event** can only occur after the **player** swaps or a **fall event**. The **grid controller** is responsible for detecting **combo events** and **fall events**. A **heads-up display** (**HUD**) keeps track of the **time** and displays a **timer** and the **score**.

Input devices all go through a single **input controller** that controls the flow of execution based on the current **state** of the **game** and the **action** bound to the **input event**.

Identification of nouns: game, player, main menu, settings menu, help menu, about menu, button, settings, options, game play area, grid, blocks, combo event, fall event, grid controller, HUD, timer, score, input device, input controller, state, action, input event

Entity classes:

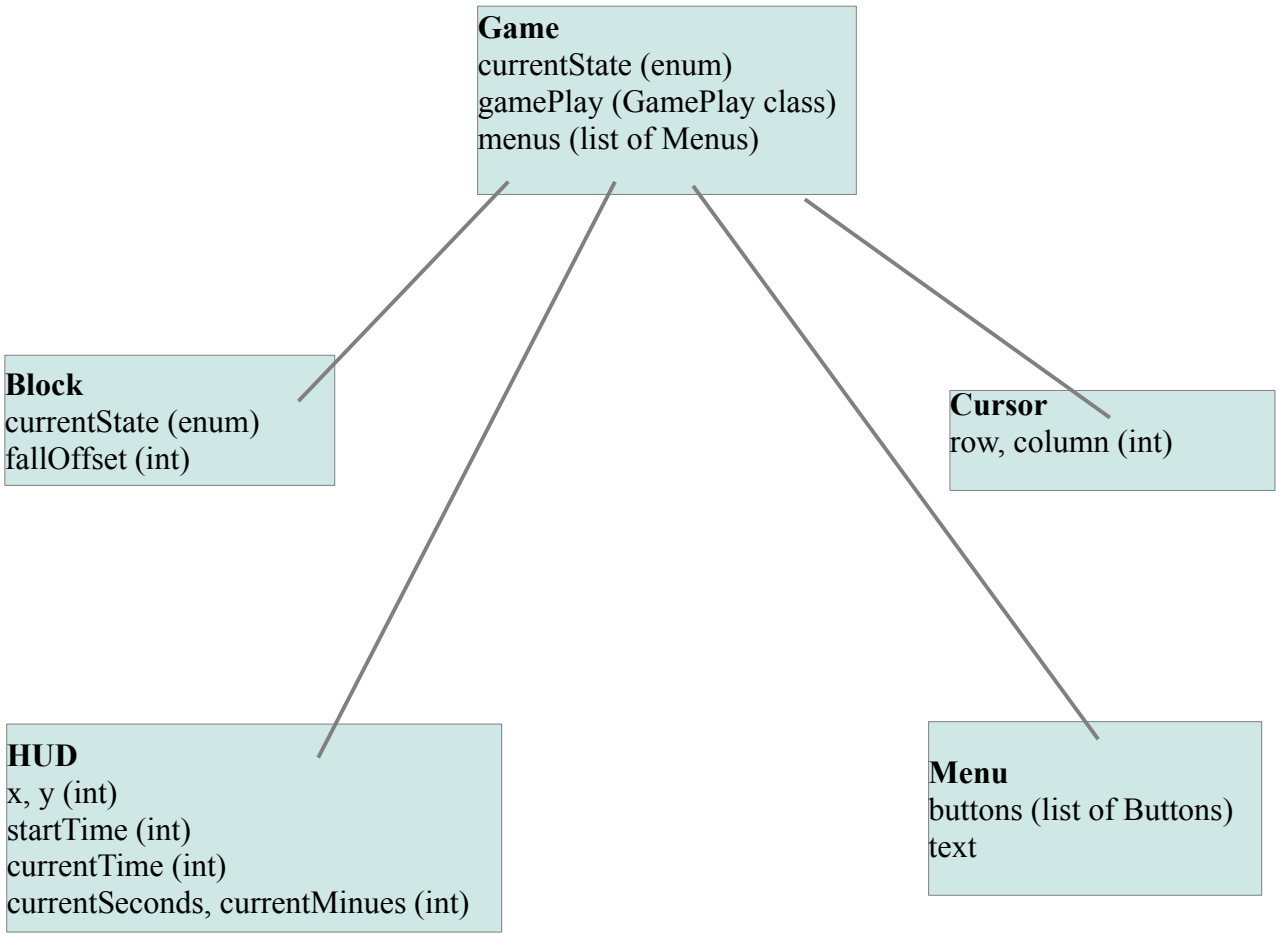
Game

HUD

Block

Cursor

Menu



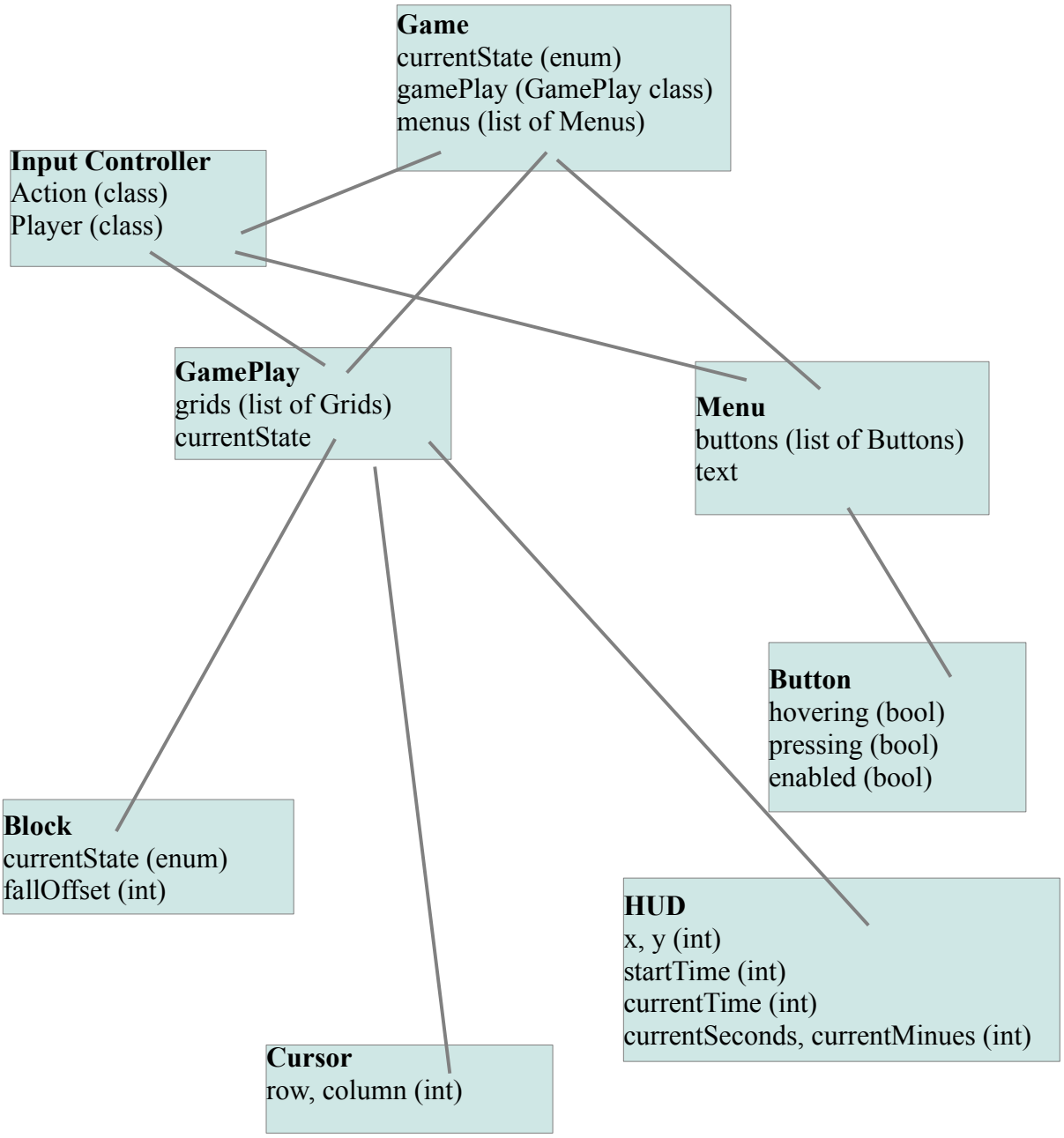
2.1.3 Boundary Class Exaction

Boundary classes:

Input

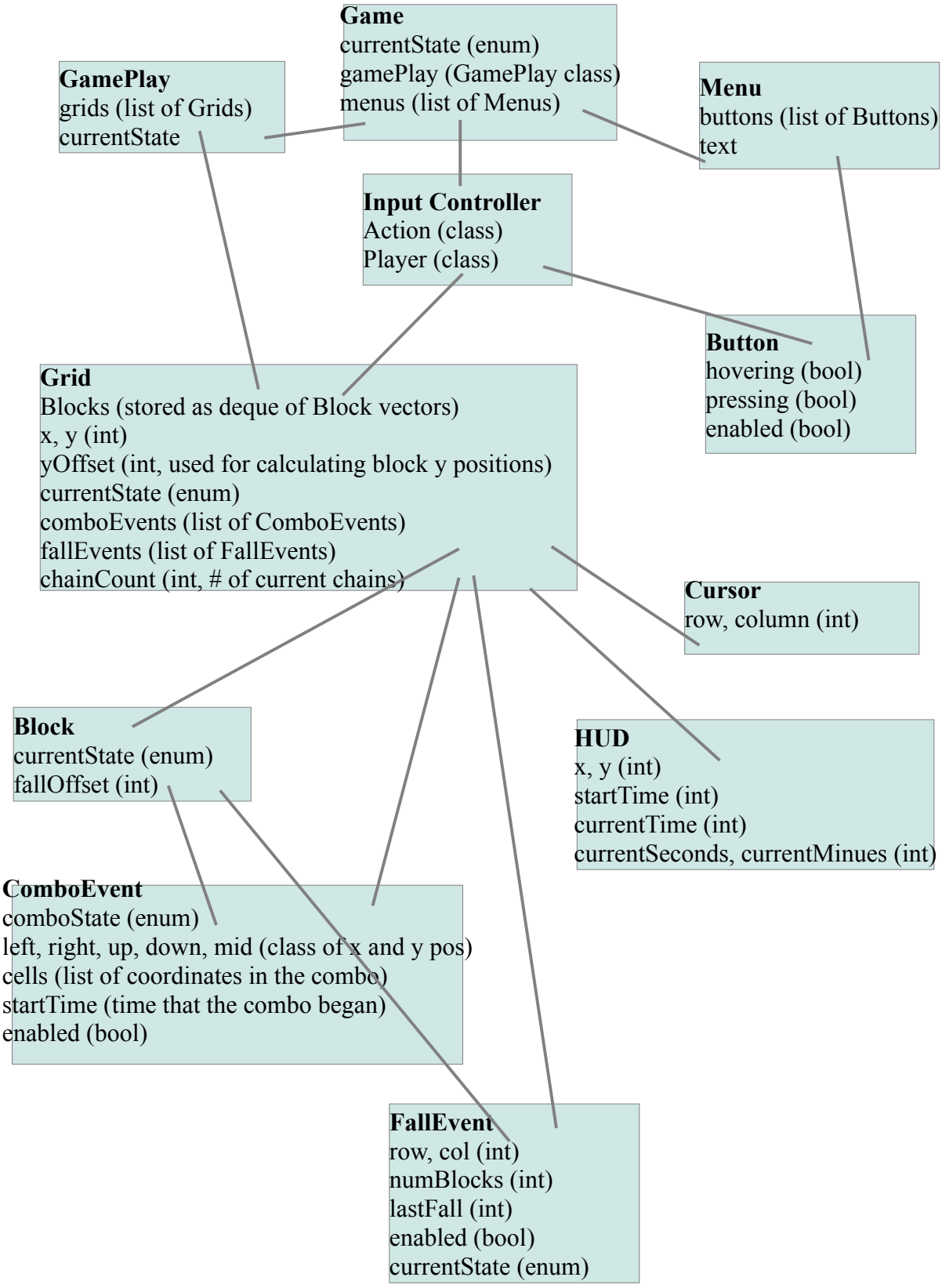
Button

GamePlay

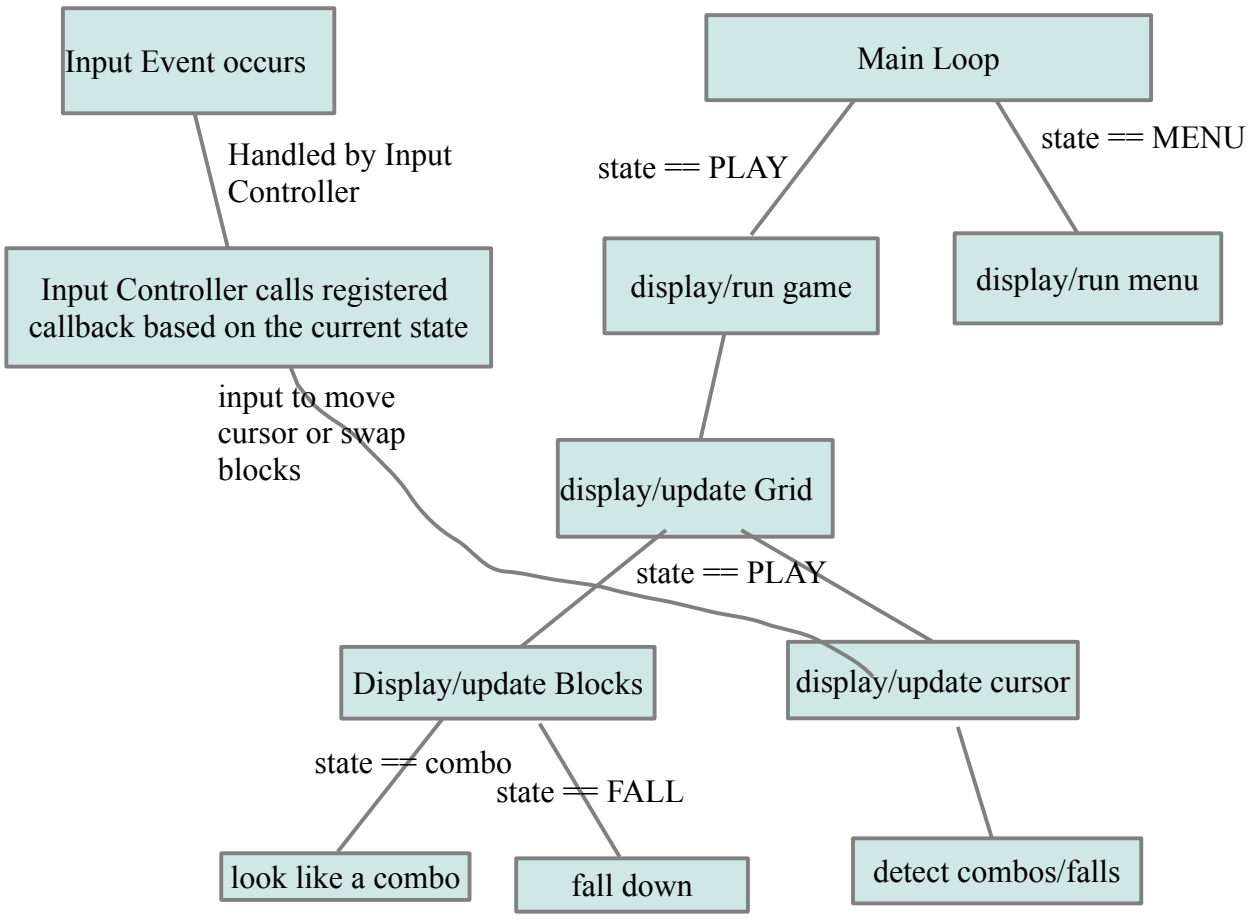


2.1.4 Control Class Modeling

Control classes: Grid, ComboEvent, FallEvent



2.1.5 Dynamic Modeling



2.1.6 Communication Modeling

2.2 Walkthrough

It seems like some attributes are missing. Re-evaluate your classes and consider what else should be added.

Make sure to include all the states in your dynamic model.

Break functional model up into different scenarios

Use UML conventions

- Walkthrough by Brandon Hinesley

2.3 Revised Analysis after Walkthrough

2.3.1 Functional Modeling

Normal Scenarios

Scenario 1:

1. Player selects Play
2. Change game's stay to PLAY
3. Initialize the Grid with randomized blocks, and initialize the timer and score.

Scenario 2:

1. Player selects Help
2. Display a helpful guide for how to play
3. Display a button to go back

Scenario 3:

1. Player selects Quit
2. Immediately exit the game

Scenario 4:

1. Player moves cursor
2. Cursor's position is recalculated based on the direction it is moved
3. Cursor is redrawn

Scenario 5:

1. Player swap blocks
2. The selected blocks are swapped with each other
3. Detect combos for each block
4. If combo is detected, goto combo detected scenario
5. Detect fall for each block
6. If fall is detected, goto fall detected scenario

Scenario 6:

1. A combo is detected
2. Increment the chain count for the combo
3. If combo has 4 or more blocks, display an animated bonus with the number of blocks in it
4. Set state of Grid and Blocks in the combo to COMBO
5. Remain in the COMBO state for (N number of blocks * T milliseconds)
6. Set the state of the COMBO blocks to DISABLED when the timer elapses, and set Grid's state to PLAY
7. Detect falling blocks
8. Group all falling blocks in the same fall event, and goto scenario for detecting a fall event

Scenario 7:

1. A fall event is detected
2. Set all the blocks in the fall event to the FALL state
3. For each block, fall Y pixels every T milliseconds while in the FALL state
4. Set block's state to ENABLED when it lands in a cell above another enabled block.
5. Detect combo for each falling block
6. If combo is detected, goto combo detected scenario

Scenario 8:

1. Player input requests to push blocks up faster
2. Grid set to PUSH state
3. Automatic block rising is temporarily disabled
4. Blocks are pushed up every T milliseconds, an interval less than the automatic interval
5. Grid restored to PLAY state when input binding is no longer active

Scenario 9:

1. A block comes into contact with the top of the Grid
2. Game set to GAMEOVER state for T milliseconds
3. Game over graphic displayed on screen while in GAMEOVER state, and player controls are disabled
4. Go back to main menu when timer elapses

Exception Scenarios

Scenario 1:

1. Player sends input that is not a registered binding
2. Nothing happens

Identification of Nouns

When the **game** loads, the **player** is presented with the **main menu**. The **player** can select **buttons** that take **actions** when selected. The **buttons** on the **main menu** are **Play**, **Help**, and **Quit**. The **Help menu** displays explains how to play the **game**. The **Quit** button closes the game. The **player** clicks the **Play button** and to start the **game**.

The **game play** consists of a **grid** for each player. A **grid** contains **blocks** and a **cursor**. The **player** controls the **cursor** and can use it to swap **blocks**. If at least 3 **blocks** are lined up, a **combo event** occurs. If there are disabled **blocks** under an enabled **block**, then a **fall event** occurs. A **fall event** can only occur after the player swaps **blocks** or after a **combo event.** A **combo event** can only occur after the **player** swaps or after a **fall event**. The **grid** is responsible for detecting **combo events** and **fall events**. An animated **bonus** is displayed when the player gets a big **combo** or a **chain**. A **heads-up display** (**HUD**) displays a **timer** and the **score**.

Input devices all go through a single **input controller** that controls the flow of execution based on the current **state** of the **game** and the **action** bound to the **input event**.

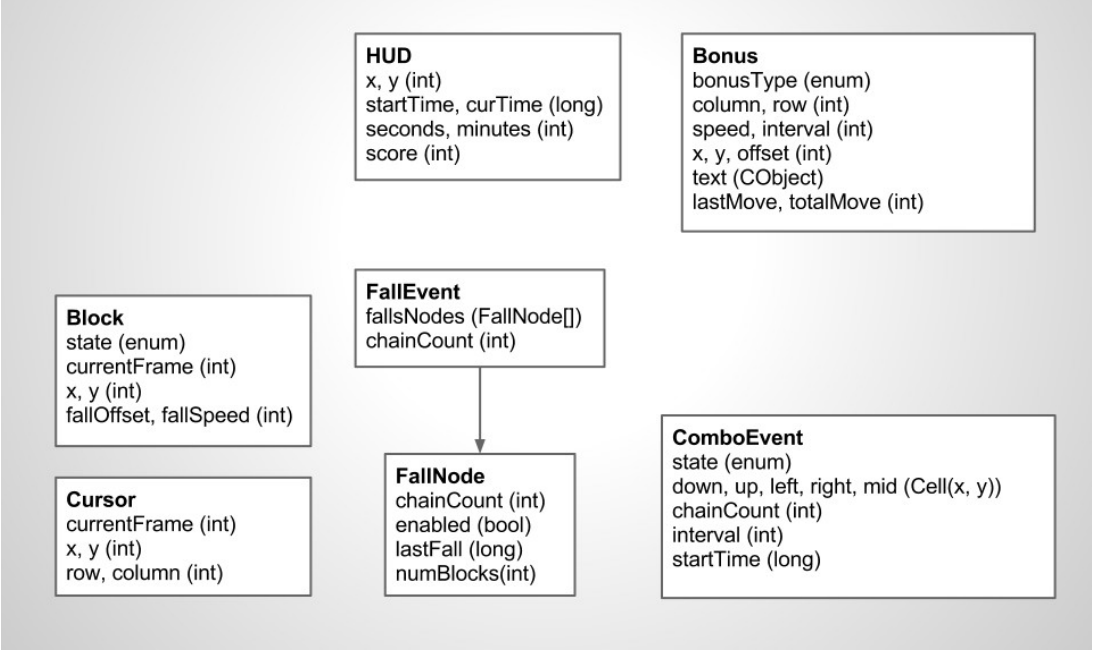
Noun Extraction

player, menu, button, actions, grid, block, cursor, combo event, fall event, bonus, HUD, timer, score, input device, input controller, state

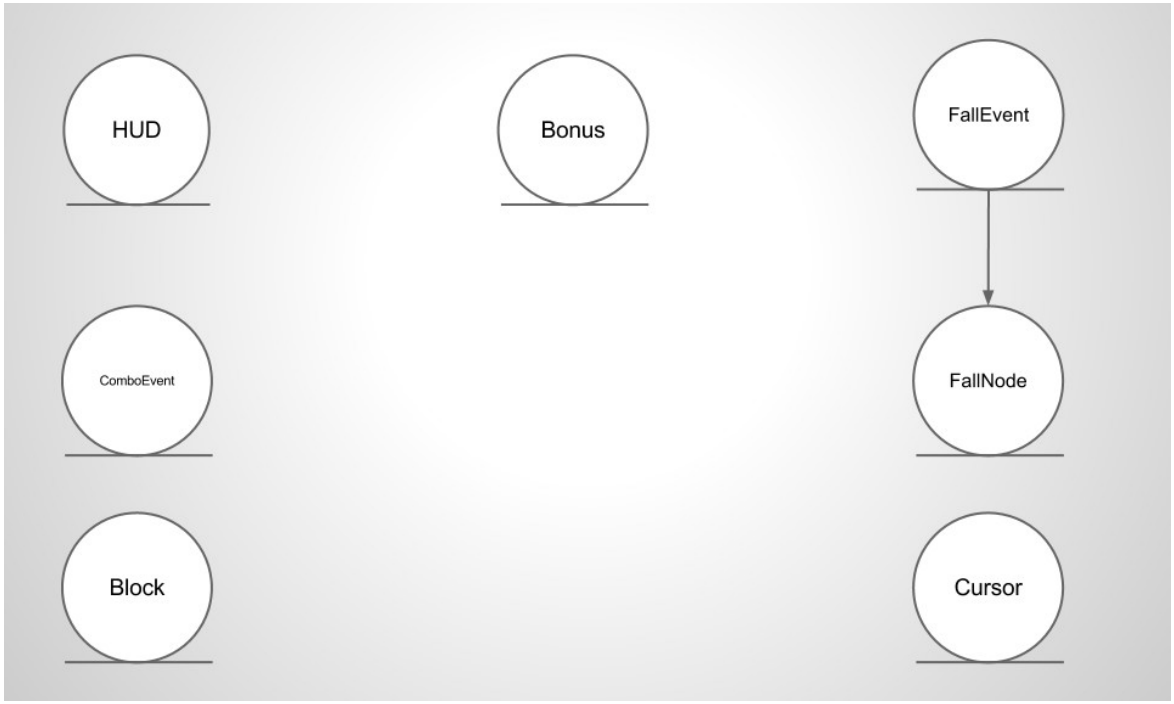
2.3.2 Entity Class Modeling

Noun Extraction

button, block, cursor, combo event, fall event, bonus, HUD, timer, score



Attribute Box Diagram

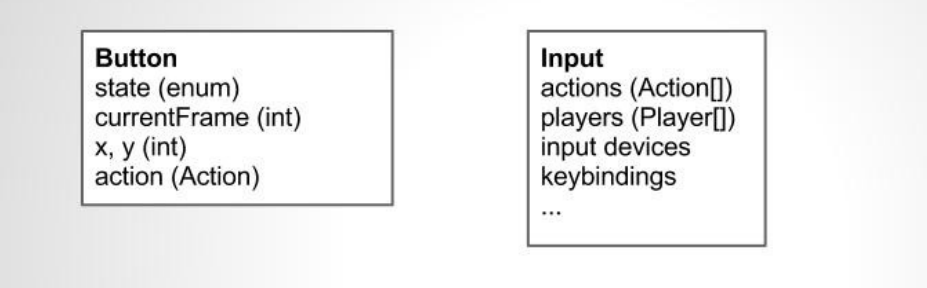


UML Class Diagram

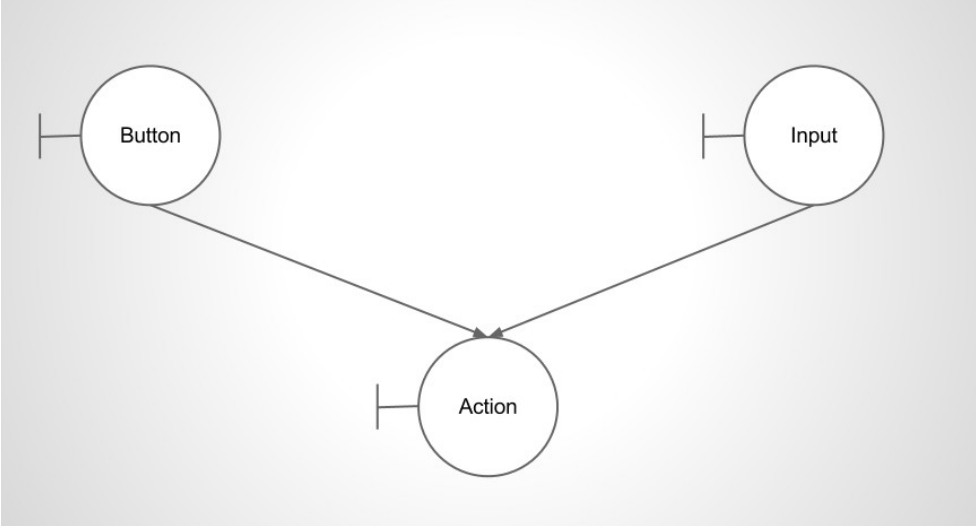
2.3.3 Boundary Class Modeling

Noun Extraction:

button, player, input, input device, action



Attribute Box Diagram

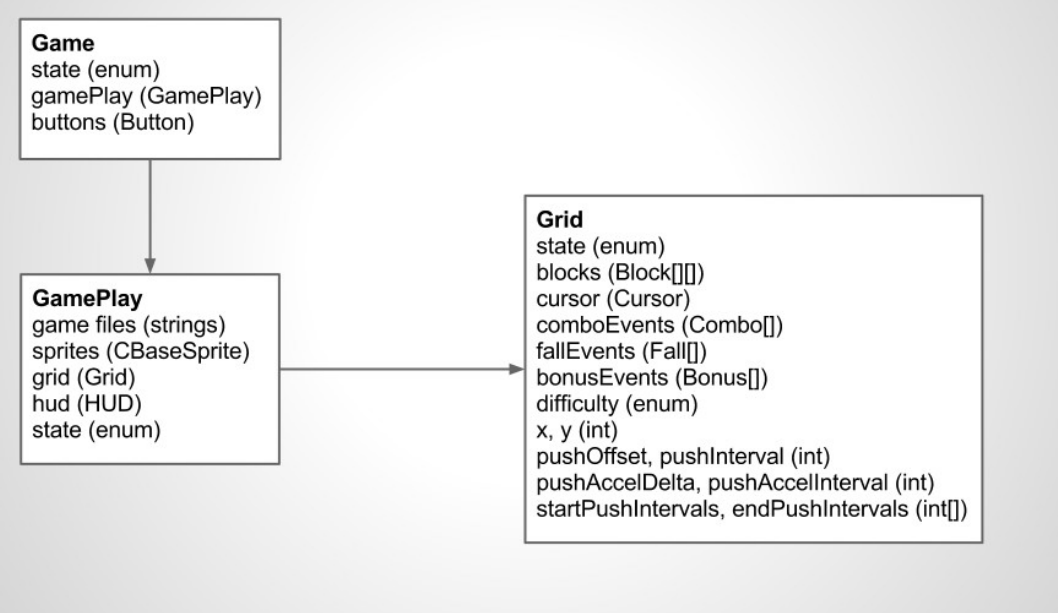


UML Class Diagram

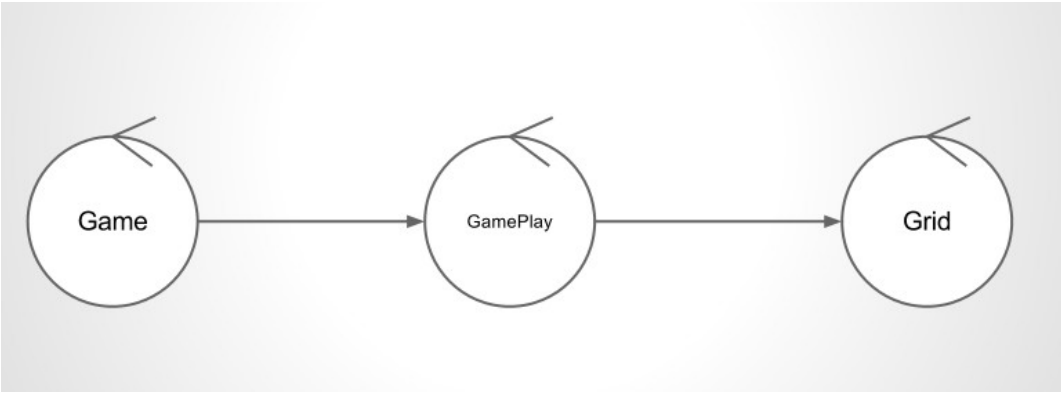
2.3.4 Control Class Modeling

Noun Extraction:

game, game play, grid

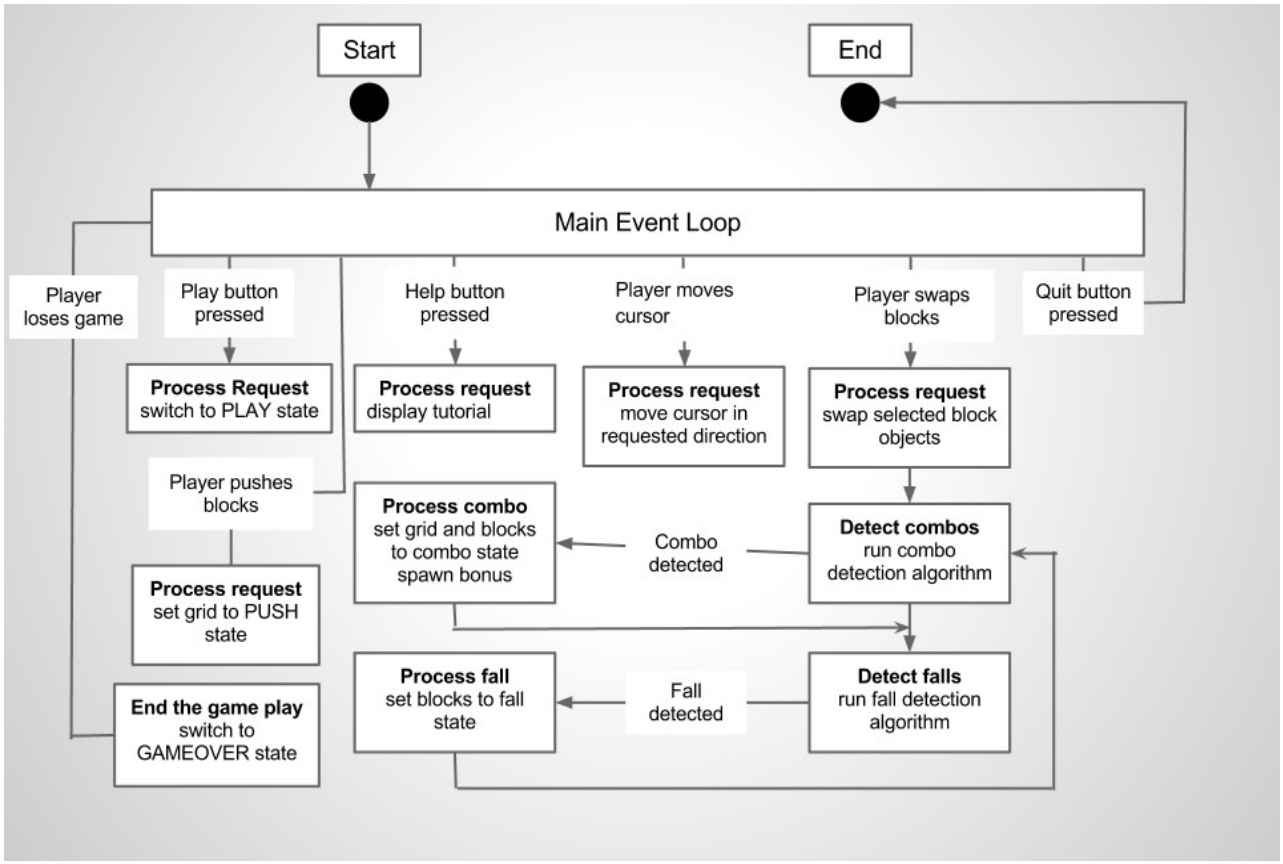


Attribute Box Diagram



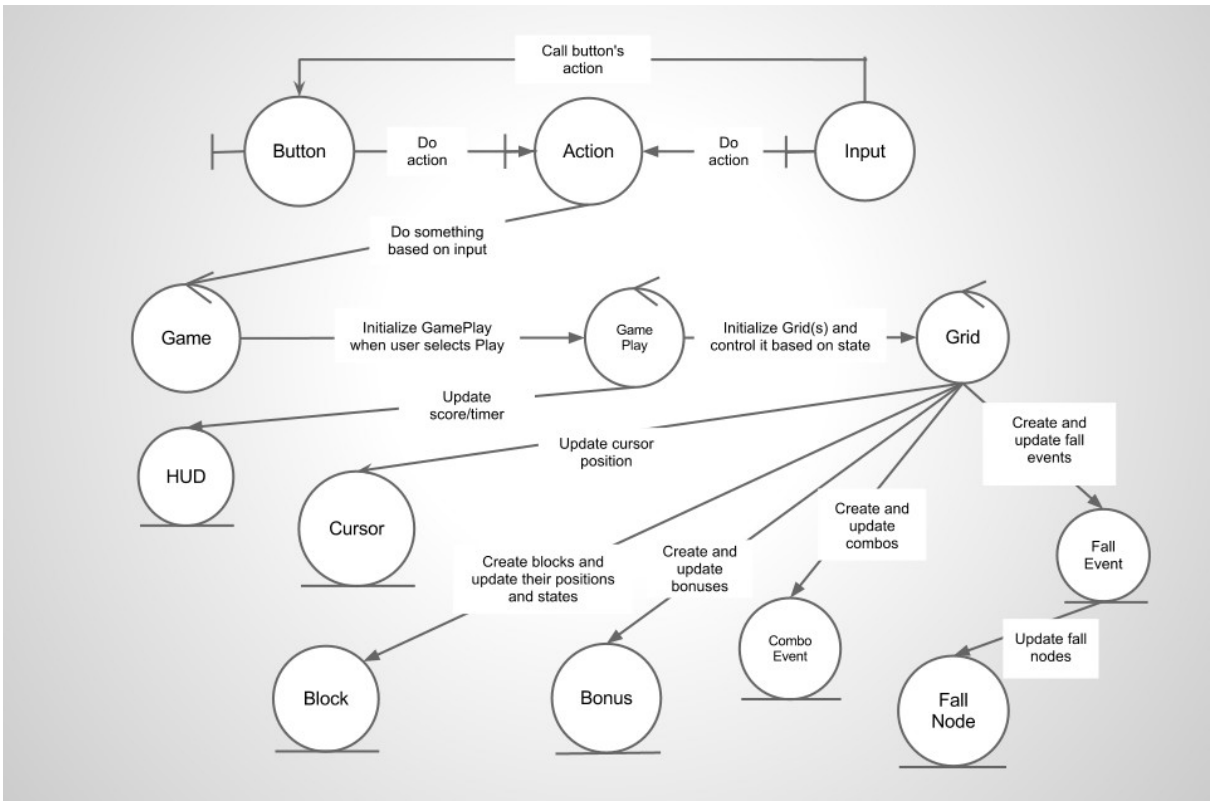
UML Class Diagram

2.3.5 Dynamic Modeling



Dynamic Model Diagram

2.3.6 Communication Modeling



Communication Model Diagram