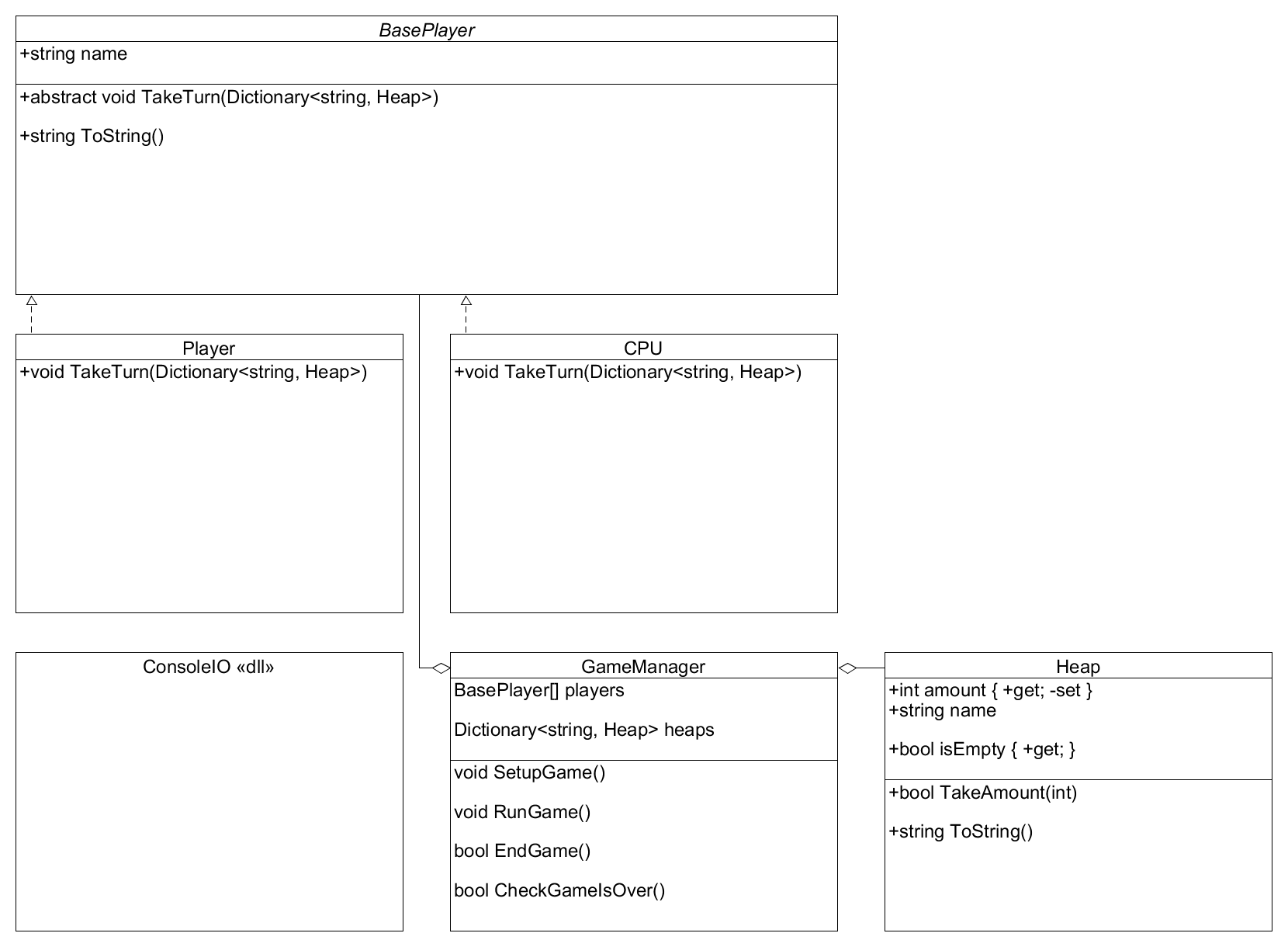
**UML**



**Class Descriptions**

**BasePlayer** – Abstract class that **Player** and **CPU** derive from.

*name* – An arbitrary string to label the player in UI interactions. Determined by the player(s) in the program.

*TakeTurn()* – An abstract function that is defined by the child classes. See **Player** and **CPU** for details.

*ToString()* – Overridden to return *name* for ease of use.

**CPU** – A player with AI determining moves.

*TakeTurn()* – Chooses a random non-empty **Heap** to take a random amount from that’s less than or equal to the **Heap’s** *amount*. If the **Heap** is the last non-empty **Heap** it instead takes one less than the **Heap’s** *amount*. If the **Heap** is the last non-empty **Heap** and it’s *amount* is one, it instead takes one.

**GameManager** – A managing class that contains the **Players**, **Heaps**, and various functions used in the game.

*players* – An array of **BasePlayer** that contains both players.

*heaps* – A Dictionary of **Heaps** that use the **Heaps** *name* as a key

*SetupGame()* – Prompts the player through the console to choose the game’s starting modes. These include if the game is **Player** vs **Player** or **Player** vs **CPU**, if the game is easy (heaps are 3, 3), medium (heaps are 2, 5, 7), or hard (2, 3, 8, 9). The players also choose names for each **Player**.

*RunGame()* – Calls *SetupGame()*. It then iterates through *players* calling *TakeTurn()* for each. After each turn is taken it calls *CheckGameIsOver()* to see if a player has won. If theyhave, *EndGame()* is called.

*EndGame()* – Prints which player won and asks the players if they would like to play again. If they answer yes, the game goes back to the beginning. If not, the program exits.

*CheckGameIsOver()* – Returns true if every **Heap** in *heaps* is empty.

**Heap** – A class that represents a pile of objects used in the game. Each turn a player chooses a pile to remove items from as their turn.

*amount* – An integer property that represents how many items are in the pile. Public getter, private setter.

*name* – A string used in the *ToString()* and in **GameManager** as the key for each **Heap** in *heaps*.

*isEmpty* – A boolean property that returns if *amount* is zero. Public getter, no setter.

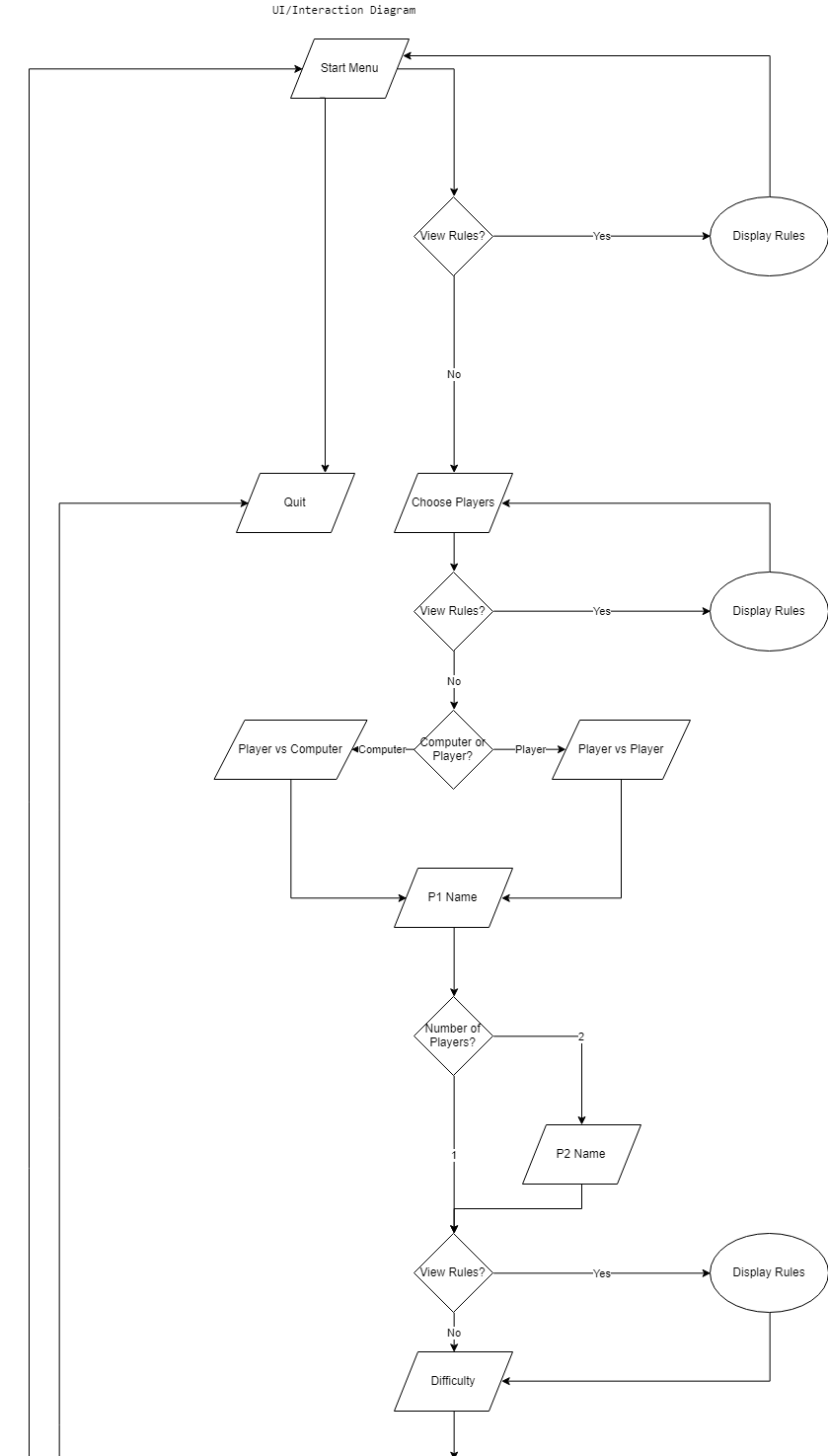
*TakeAmount()* – Checks if the integer passed is less than or equal to *amount*. If it is, it is subtracted from *amount* and *TakeAmount()* returns true. Otherwise *TakeAmount()* returns false.

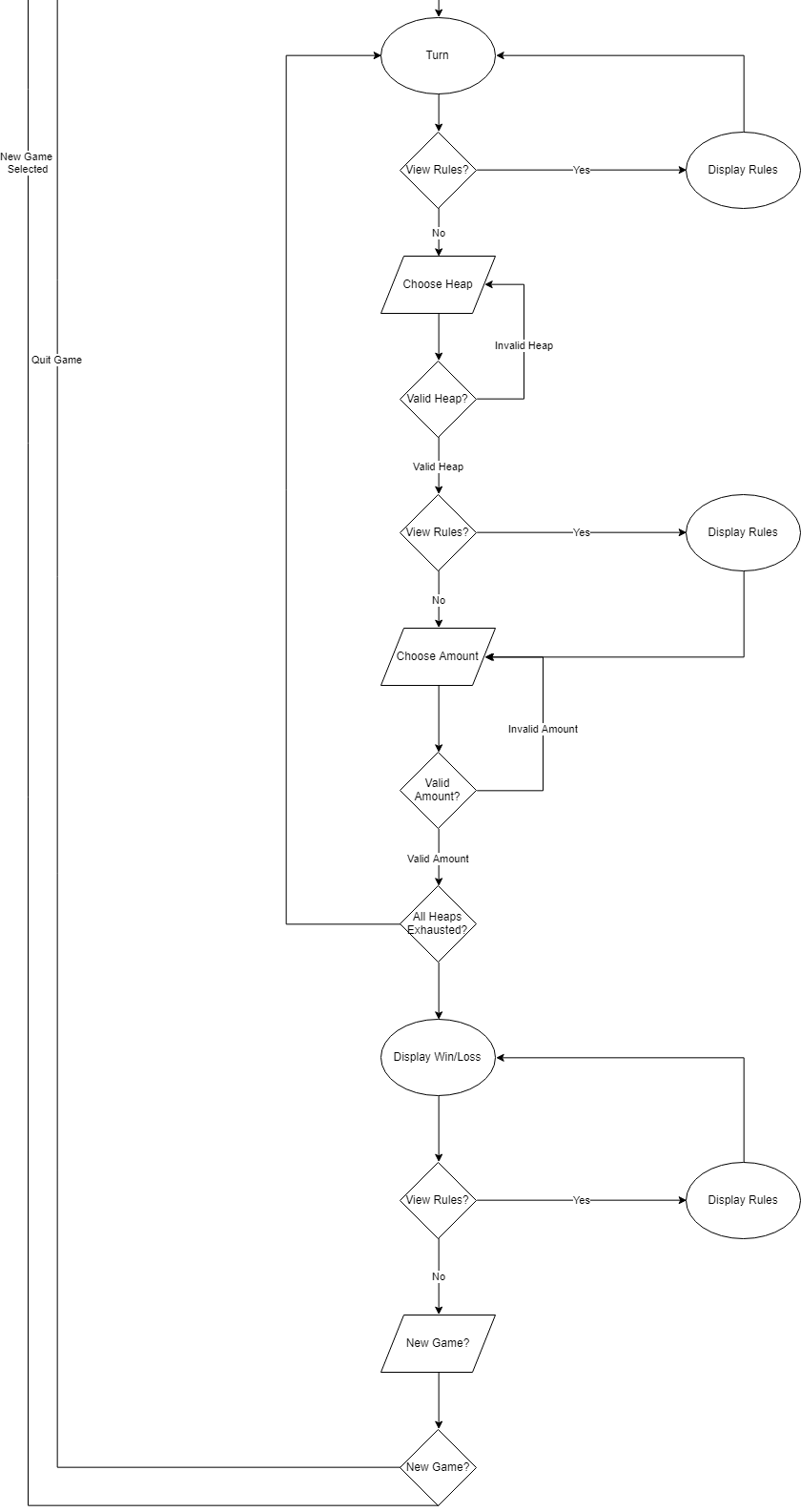
*ToString()* – Returns *name* concatenated to “ - “ concatenated to *amount*.

**Player** – A player with a human actor making decisions via UI.

*TakeTurn()* – Prompts the actor to choose a **Heap** to take items from. Tells the actor they chose an invalid **Heap** until a **Heap** that exists and isn’t empty is chosen. Prompts the actor to choose an amount to remove from that pile. Tells the actor they did not type a number until they typed a number. Tells the actor they cannot take that much until they enter a number less than or equal to the **Heap’s** *amount*.

**Flow Diagram**

****



**UI Templating**

Start Menu

|  |
| --- |
| Welcome to the game of Nim  1) Select Players  2) Quit |

Quit

|  |
| --- |
| Thanks for playing |

Player Select

|  |
| --- |
| Select Players  1) Player vs. Player  2) Player vs. Computer  3) Quit |

Name Selection

|  |
| --- |
| Enter playern’s name: \_\_\_\_\_\_\_\_ |

Choose Difficulty

|  |
| --- |
| Pick a difficulty  1) Easy (3 3)  2) Medium (2 5 7)  3) Hard (2 3 8 9) |

Start of Turn

|  |
| --- |
| Playern’s Turn  1) Choose pile  2) Quit |

Turn

|  |
| --- |
| Pile A: N  Pile B: N  Pile C: N  Pile D: N  Choose a pile:\_  Pick amount to take:\_\_\_ |

Win

|  |
| --- |
| Playern wins  Do you want to play again?  1) yes  2) no |