# ECE 421 Assignment 4 Part 3

**Winter2019_Group4**:

Nathan Klapstein (1449872)

Tony Qian (1396109)

Thomas Lorincz (1461567)

Zach Drever (1446384)

---

## Table of Contents

# Summary of Functional Deviations

## AppModel

### CPU vs CPU Mode

In the course of building the CPU player, we have selected not to have a CPU vs. CPU player mode as an option in the main app menu. While this game type is possible to implement with our architecture, it did not seem useful to include it because it does not demo well (game is played in an instant) and this mode did not help our contract-based test cases.

## Computer Opponent

### Component of AppModel

Our AppModel integrates CPU player capabilities in a extensible manner that is easily added to our game. We chose to include the CPU as part of the AppModel because the game logic was also included in the AppModel. Having our CPU player close to the game logic made building the AI more intuitive to build.

### Variable Difficulty

Our AI player works with a simple yet effective strategy. It first tries to find a move which may win the game, and plays that if possible. Otherwise, it will try to find a move in which the enemy player may win the game, and block that if possible. Lastly, if it was not able to calculate either effective moves, our AI finds the next best move which may help them win the game earliest, and plays that. Our app implements one difficulty as it is representative of how the majority of people would play Connect 4. We chose to spend time trying to build a challenging AI opponent rather than variable difficulties, and time constraints stopped us from adding the variable difficulty functionality later.

## Game Board

### Support for a CLI

We now also support the ability to play Connect 4 and Toot and Otto on the command line. The decision to build out this functionality was made because of our use of the MVP design pattern. The CLI could be treated as a new View that will subscribe to the same observer pattern that the GUI-based View does. In this way, the addition of a CLI took very little time (2 hours) because our architecture choice is highly maintainable.

# Concrete Test Cases

In part 2, we listed contracts that we would subscribe to through our unit tests. Below are the unit test implementations. Please note: Many of our preconditions are asserted by our user interface design. For example, it may be a precondition that we cannot be in a game mode type other than “Connect 4” or “Toot and Otto.” Our UIs only allow these choices, so there is no need to validate the precondition.

```
def test_update_turn
 assert_equal(AppModel::PLAYER_1_TURN, @model.state[:turn])
 @model.update_turn(AppModel::PLAYER_2_TURN)
 assert_equal(AppModel::PLAYER_2_TURN, @model.state[:turn])
end

def test_update_type
 assert_equal(AppModel::CONNECT_4, @model.state[:type])
 @model.update_game_type(AppModel::TOOT_AND_OTTO)
 assert_equal(AppModel::TOOT_AND_OTTO, @model.state[:type])
end

def test_update_mode
 assert_equal(AppModel::PLAYER_PLAYER, @model.state[:mode])
 @model.update_game_mode(AppModel::PLAYER_CPU)
 assert_equal(AppModel::PLAYER_CPU, @model.state[:mode])
 @model.update_game_mode(AppModel::CPU_PLAYER)
 assert_equal(AppModel::CPU_PLAYER, @model.state[:mode])
end

def test_update_phase
 assert_equal(AppModel::MENU, @model.state[:phase])
 @model.update_game_phase(AppModel::IN_PROGRESS)
 assert_equal(AppModel::IN_PROGRESS, @model.state[:phase])
 @model.update_game_phase(AppModel::GAME_OVER)
 assert_equal(AppModel::GAME_OVER, @model.state[:phase])
end

def test_column_full
 6.times do
  @model.place_token(0)
 end
 assert_equal(AppModel::PLAYER_1_TURN, @model.state[:turn])
 @model.place_token(0)
 assert_equal(AppModel::PLAYER_1_TURN, @model.state[:turn])
end
```

```
def test_c4_vert
 3.times do
   @model.place_token(0)
   @model.place_token(1)
 end
 @model.board_place_token(0)
 assert_true(@model.connect_4_vertical?)
end

def test_c4_horiz
 3.times do |i|
   @model.place_token(i)
   @model.place_token(i)
 end
 @model.board_place_token(3)
 assert_true(@model.connect_4_horizontal?)
end

def test_c4_right_diag
 @model.place_token(0) #1
 @model.place_token(1) #2
 @model.place_token(1) #1
 @model.place_token(2) #2
 @model.place_token(3) #1
 @model.place_token(2) #2
 @model.place_token(2) #1
 @model.place_token(3) #2
 @model.place_token(3) #1
 @model.place_token(4) #2
 @model.board_place_token(3)
 assert_true(@model.connect_4_right_diagonal?)
end

def test_c4_left_diag
 @model.place_token(0) #1
 @model.place_token(0) #2
 @model.place_token(0) #1
 @model.place_token(1) #2
 @model.place_token(0) #1
 @model.place_token(1) #2
 @model.place_token(1) #1
 @model.place_token(2) #2
 @model.place_token(2) #1
 @model.place_token(4) #2
 @model.board_place_token(3) #1
 assert_true(@model.connect_4_left_diagonal?)
end
```

```
def test_ot_horiz
 @model.place_token(0)
 @model.place_token(1)
 @model.place_token(3)
 @model.board_place_token(2)
 assert_equal(AppModel::PLAYER_1_WINS, @model.toot_and_otto_horizontal)
end

def test_ot_vert
 @model.place_token(0)
 @model.place_token(0)
 @model.place_token(1)
 @model.place_token(0)
 @model.board_place_token(0)
 assert_equal(AppModel::PLAYER_1_WINS, @model.toot_and_otto_vertical)
end

def test_ot_right_diag
 @model.place_token(0) #1
 @model.place_token(2) #2
 @model.place_token(1) #1
 @model.place_token(1) #2
 @model.place_token(2) #1
 @model.place_token(2) #2
 @model.place_token(3) #1
 @model.place_token(3) #2
 @model.place_token(3) #1
 @model.place_token(4) #2
 @model.board_place_token(3)
 assert_equal(AppModel::PLAYER_1_WINS, @model.toot_and_otto_right_diagonal)
end

def test_ot_left_diag
 @model.place_token(0) #1
 @model.place_token(0) #2
 @model.place_token(0) #1
 @model.place_token(1) #2
 @model.place_token(0) #1
 @model.place_token(1) #2
 @model.place_token(2) #1
 @model.place_token(1) #2
 @model.place_token(3) #1
 @model.board_place_token(2)
 assert_equal(AppModel::PLAYER_1_WINS, @model.toot_and_otto_left_diagonal)
end
```