Kuncheng Feng
CSC 466 presentation

Updated play function

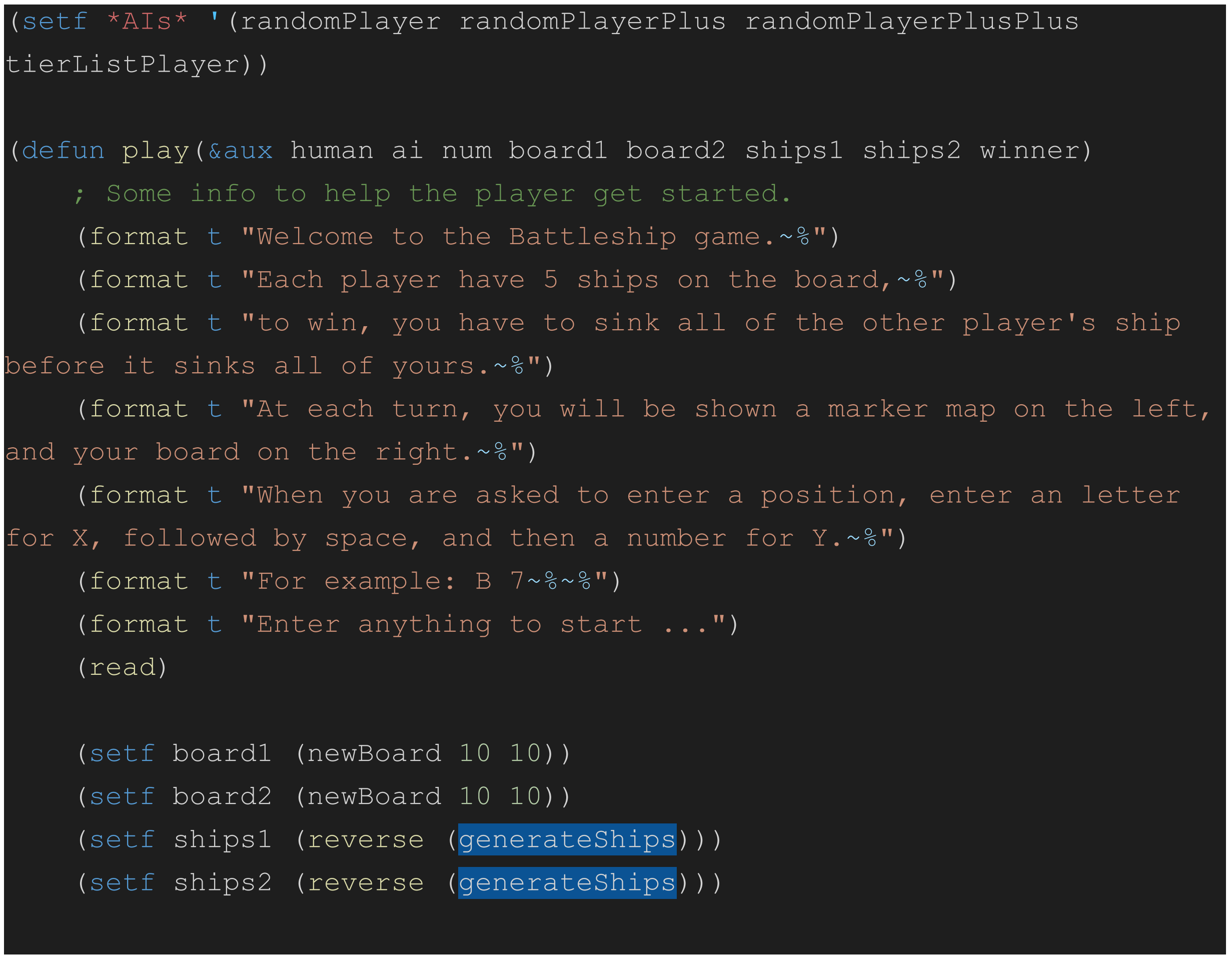
Abstract

This is a simple update to the play function, (human) player can now choose which AI he wants to play against.

Code

The main play function first displays some information to let the user know what to input, then it creates an opponent and has the user play with it until the game ends.

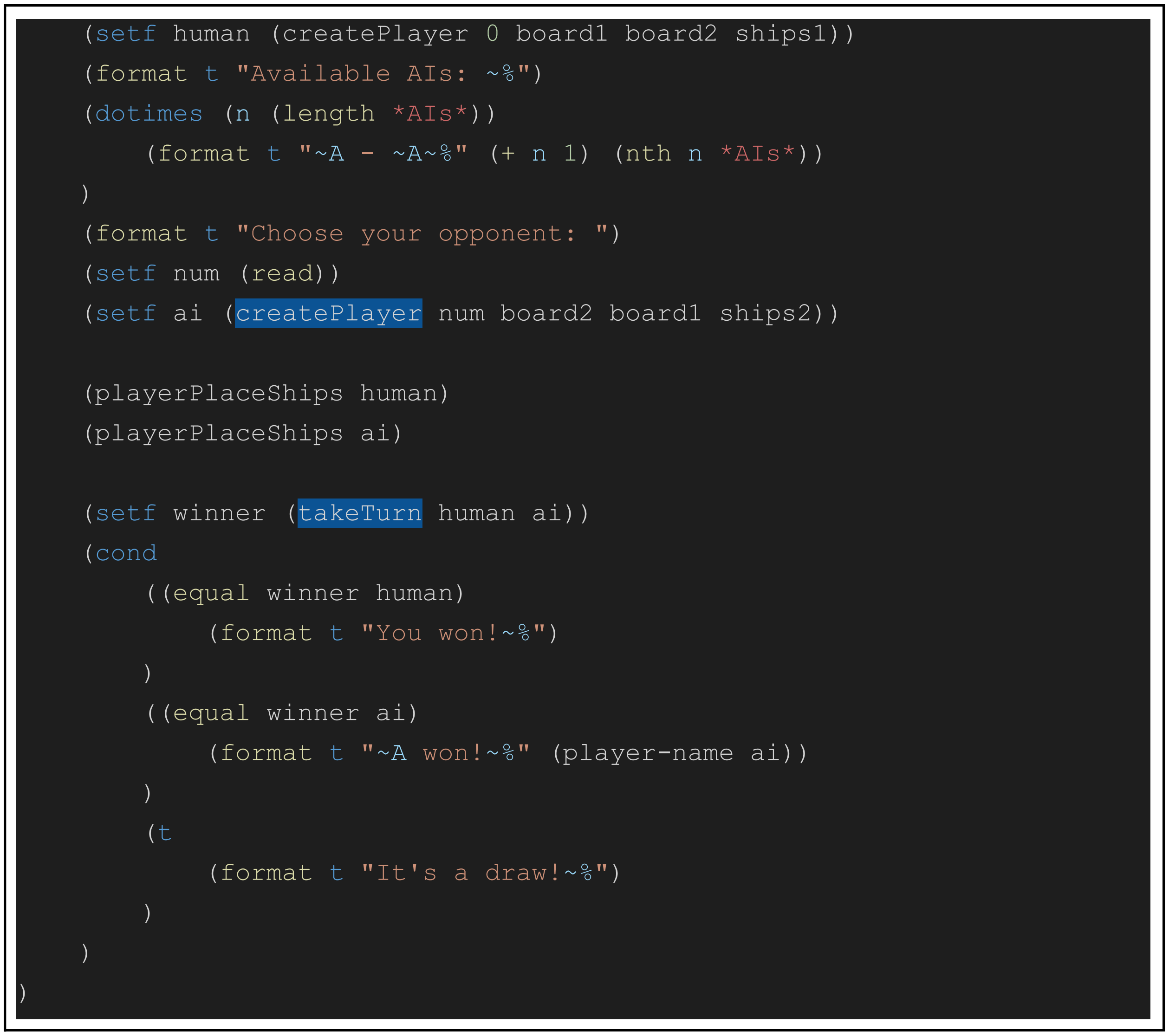
I have also highlighted the functions that will be shown in further detail.



```
(setf *AIs* '(randomPlayer randomPlayerPlus randomPlayerPlusPlus
tierListPlayer))

(defun play(&aux human ai num board1 board2 ships1 ships2 winner)
    ; Some info to help the player get started.
    (format t "Welcome to the Battleship game.~%")
    (format t "Each player have 5 ships on the board,~%")
    (format t "to win, you have to sink all of the other player's ship
before it sinks all of yours.~%")
    (format t "At each turn, you will be shown a marker map on the left,
and your board on the right.~%")
    (format t "When you are asked to enter a position, enter an letter
for X, followed by space, and then a number for Y.~%")
    (format t "For example: B 7~%~%")
    (format t "Enter anything to start ...")
    (read)

    (setf board1 (newBoard 10 10))
    (setf board2 (newBoard 10 10))
    (setf ships1 (reverse (generateShips)))
    (setf ships2 (reverse (generateShips)))
```

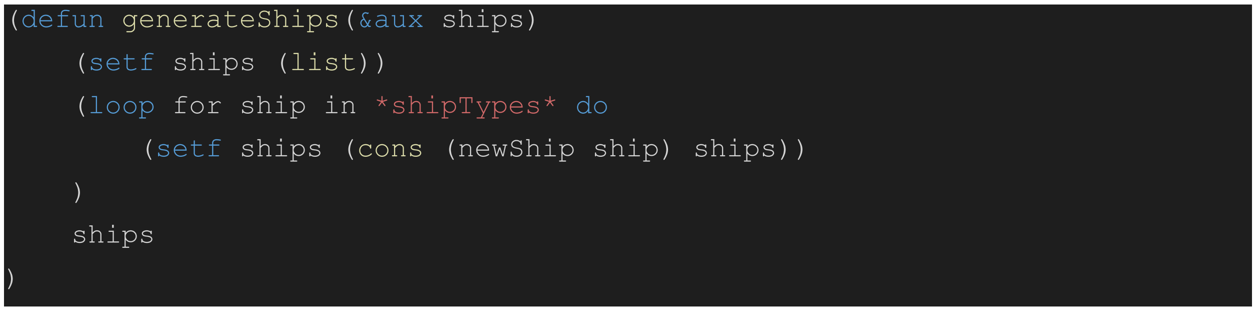


```
    (setf human (createPlayer 0 board1 board2 ships1))
    (format t "Available AIs: ~%")
    (dotimes (n (length *AIs*))
        (format t "~A - ~A~%" (+ n 1) (nth n *AIs*))
    )
    (format t "Choose your opponent: ")
    (setf num (read))
    (setf ai (createPlayer num board2 board1 ships2))

    (playerPlaceShips human)
    (playerPlaceShips ai)

    (setf winner (takeTurn human ai))
    (cond
        ((equal winner human)
            (format t "You won!~%")
        )
        ((equal winner ai)
            (format t "~A won!~%" (player-name ai))
        )
        (t
            (format t "It's a draw!~%")
        )
    )
)
```

The “generateShip” function returns a list of ship instances, each of a different type:



```
(defun generateShips(&aux ships)
    (setf ships (list))
    (loop for ship in *shipTypes* do
        (setf ships (cons (newShip ship) ships))
    )
    ships
)
```

The “createPlayer” function takes in a number and the necessary game objects to create a corresponding player instance:

```lisp
; Return a player instance.
(defun createPlayer(num (this board) (other board) (ships list))
    (cond
        ; 0 = human player
        ((equal num 0)
            (newHumanPlayer this other ships)
        )
        ; 1 = random player
        ((equal num 1)
            (newRandomPlayer this other ships)
        )
        ; 2 = random player +
        ((equal num 2)
            (newRandomPlayerPlus this other ships)
        )
        ; 3 = random player ++
        ((equal num 3)
            (newRandomPlayerPlusPlus this other ships)
        )
        ; 4 = tier list player
        ((equal num 4)
            (newTierListPlayer this other ships)
        )
        (t
            (format t "No player correspond to the number entered: ~A~%"
num)
            (bye)
        )
    )
)
```

The “takeTurn” function is a self recursive function that continues until the game ends, and returns the winner’s player instance, or nil if it's a draw.

```lisp
; Return the victor player's instance.
(defun takeTurn(player1 player2 &aux p1Win p2Win)
    (playerOpenFire player1)
    (playerOpenFire player2)

    (setf p1Win (isPlayerDefeated player2))
    (setf p2Win (isPlayerDefeated player1))

    (cond
        ((and p1Win p2Win)
            nil
        )
        (p1Win
            player1
        )
        (p2Win
            player2
        )
        (t
            (takeTurn player1 player2)
        )
    )
)
```

Demo

No demo is present in this pdf, as they will be shown repeatedly when a new AI player is introduced.