A) Old isWinningMove Function (slgdifferent heuristics)

```
(define/public (isWinningMove col player-num)
    (define temp (copy-2d vec-board height width))
    (begin
      (2d-vector-set! temp 0 col (+ 1 (2d-vector-ref temp 0 col)))
      (2d-vector-set! temp (- height (2d-vector-ref temp 0 col)) col player-num))
    (define i (- height (2d-vector-ref temp 0 col)))
    (if (canPlay col) (cond [(check-horizontal temp i col player-num col col) #t]
        [(check-diag1 temp i col player-num (cons i col) (cons i col)) #t]
        [(check-diag2 temp i col player-num (cons i col) (cons i col)) #t]
        [(check-vertical temp i col player-num i i) #t]
        [else #f]) #f))

  (define (check-horizontal vec r c player-num left right)
    (cond [(>= (- right left) 3) #t]
        [(and (>= (- left 1) 0) (< (+ right 1) width) (= (2d-vector-ref vec r (- left 1)) player-num) (=
(2d-vector-ref vec r (+ right 1)) player-num))
                                        (check-horizontal vec r c player-num (- left 1) (+ right 1))]
        [(and (>= (- left 1) 0) (= (2d-vector-ref vec r (- left 1)) player-num))
                    (check-horizontal vec r c player-num (- left 1) right)]
        [(and (< (+ 1 right) width) (= (2d-vector-ref vec r (+ right 1)) player-num))
                        (check-horizontal vec r c player-num left (+ 1 right))]
        [else #f]))

  (define (check-vertical vec r c player-num top bottom)
    (cond [(>= (- bottom top) 3) #t]
        [(and (> (- top 1) 0) (< (+ bottom 1) height) (= (2d-vector-ref vec (- top 1) c) player-num)
(= (2d-vector-ref vec (+ bottom 1) c) player-num))
                                        (check-vertical vec r c player-num (- top 1) (+ bottom 1))]
        [(and (> (- top 1) 0) (= (2d-vector-ref vec (- top 1) c) player-num))
                    (check-vertical vec r c player-num (- top 1) bottom)]
        [(and (< (+ 1 bottom) width) (= (2d-vector-ref vec (+ 1 bottom) c) player-num))
                        (check-vertical vec r c player-num top (+ 1 bottom))]
        [else #f]))

  (define (check-diag1 vec r c player-num top-left bottom-right)
    (cond [(>= (- (car bottom-right) (car top-left)) 3) #t]
        [(and (> (- (car top-left) 1) 0) (>= (- (cdr top-left) 1) 0) (< (+ (car bottom-right) 1) height) (<
(+ (cdr bottom-right) 1) width) (= (2d-vector-ref vec (- (car top-left) 1) (- (cdr top-left) 1))
player-num)
                                        (= (2d-vector-ref vec (+ (car bottom-right) 1) (+ (cdr
bottom-right) 1)) player-num))
```

```
                                        (check-diag1 vec r c player-num (cons (- (car top-left) 1)
(- (cdr top-left) 1))
                                                (cons (+ 1 (car bottom-right)) (+ 1 (cdr
bottom-right))))]
          [(and (> (- (car top-left) 1) 0) (>= (- (cdr top-left) 1) 0)
                                        (= (2d-vector-ref vec (- (car top-left) 1) (- (cdr top-left) 1))
player-num))
                                        (check-diag1 vec r c player-num (cons (- (car top-left) 1)
(- (cdr top-left) 1)) bottom-right)]
          [(and (< (+ (car bottom-right) 1) height) (< (+ (cdr bottom-right) 1) width)
                                        (= (2d-vector-ref vec (+ (car bottom-right) 1) (+ (cdr
bottom-right) 1)) player-num))
                                        (check-diag1 vec r c player-num top-left (cons (+ 1 (car
bottom-right)) (+ 1 (cdr bottom-right))))]
          [else #f]))

    (define (check-diag2 vec r c player-num top-right bottom-left)
      (cond [(>= (- (car bottom-left) (car top-right)) 3) #t]
          [(and (< (+ (car bottom-left) 1) height) (>= (- (cdr bottom-left) 1) 0) (> (- (car top-right) 1)
0) (< (+ (cdr top-right) 1) width)
                                        (= (2d-vector-ref vec (+ (car bottom-left) 1) (- (cdr
bottom-left) 1)) player-num)
                                        (= (2d-vector-ref vec (- (car top-right) 1) (+ (cdr top-right)
1)) player-num))
                                        (check-diag2 vec r c player-num (cons (- (car top-right) 1)
(+ (cdr top-right) 1))
                                                (cons (+ (car bottom-left) 1) (- (cdr bottom-left)
1)))]
          [(and (< (+ (car bottom-left) 1) height) (>= (- (cdr bottom-left) 1) 0)
                                        (= (2d-vector-ref vec (+ (car bottom-left) 1) (- (cdr
bottom-left) 1)) player-num))
                                        (check-diag2 vec r c player-num top-right (cons (+ (car
bottom-left) 1) (- (cdr bottom-left) 1)))]
          [(and (> (- (car top-right) 1) 0) (< (+ (cdr top-right) 1) width)
                                        (= (2d-vector-ref vec (- (car top-right) 1) (+ (cdr top-right)
1)) player-num))
                                        (check-diag2 vec r c player-num (cons (- (car top-right) 1)
(+ (cdr top-right) 1)) bottom-left)]
          [else #f]))
```

New isWinningMove Function

```
(define/public (isWinningMove col player)
    (define h (- height (2d-vector-ref vec-board 0 col)))
    (define x (- h 1))
    (define y col)
    (cond [(canPlay col) (begin
                    (define b1 (vertical x y player))
                    (define b2 (abstract_four x y player 0 1))
                    (define b3 (abstract_four x y player 1 1))
                    (define b4 (abstract_four x y player 1 -1))
                    (if (or b1 b2 b3 b4) #t
                        #f))]
          [#t #f]))
```

B) Old Heuristic Function

```
(define/public heuristic
     (lambda (player)
       (define score1 (+ (vertical player) (horizontal player) (diag1 player) (diag2 player)))
       (define score2 (+ (vertical (- 3 player)) (horizontal (- 3 player)) (diag1 (- 3 player)) (diag2 (-
3 player))))
       (- score1 score2)))

   (define (vertical player)
     (define count 0)
     (define x 0)
     (for (set! x 0) : (< x 7) : (set! x (+ x 1)) :
       (define h (- height (2d-vector-ref vec-board 0 x)))
       (cond [(and (<= h (- height 3)) ( = (2d-vector-ref vec-board h x) player)
                ( = (2d-vector-ref vec-board (+ h 1) x) player)
                ( = (2d-vector-ref vec-board (+ h 2) x) player)) (set! count (+ count 1))]))
     count)

   (define (horizontal player)
     (define count 0)
     (define x 0)
     (define y 0)
     (begin
       (for (set! x 1) : (< x 7) : (set! x (+ x 1)) :
         (for (set! y 0) : (< y 7) : (set! y (+ y 1)) :
           (cond [(and (= (2d-vector-ref vec-board x y) 0)
                       (and (>= y 1) (= (2d-vector-ref vec-board x (- y 1)) player))
                       (and (< y (- width 1)) (= (2d-vector-ref vec-board x (+ y 1)) player))
                       (and (< y (- width 2)) (= (2d-vector-ref vec-board x (+ y 2)) player))) (set! count (+
count 1))]
                 [(and (= (2d-vector-ref vec-board x y) 0)
                       (and (>= y 1) (= (2d-vector-ref vec-board x (- y 1)) player))
                       (and (< y (- width 1)) (= (2d-vector-ref vec-board x (+ y 1)) player))
                       (and (>= y 2) (= (2d-vector-ref vec-board x (- y 2)) player))) (set! count (+ count
1))]
                 [(and (= (2d-vector-ref vec-board x y) 0)
                       (and (>= y 1) (= (2d-vector-ref vec-board x (- y 1)) player))
                       (and (>= y 3) (= (2d-vector-ref vec-board x (- y 2)) player)
                            (= (2d-vector-ref vec-board x (- y 3)) player))) (set! count (+ count 1))]
                 [(and (= (2d-vector-ref vec-board x y) 0)
                       (and (< y (- width 1)) (= (2d-vector-ref vec-board x (+ y 1)) player))
```

```
                (and (< y (- width 3)) (= (2d-vector-ref vec-board x (+ y 2)) player)
                    (= (2d-vector-ref vec-board x (+ y 3)) player))) (set! count (+ count 1))])))
    count))


  (define (diag1 player)
    (define count 0)
    (define x 0)
    (define y 0)
    (begin
      (for (set! x 1) : (< x 7) : (set! x (+ x 1)) :
        (for (set! y 0) : (< y 7) : (set! y (+ y 1)) :
          (cond [(and (= (2d-vector-ref vec-board x y) 0)
                (and (> x 1) (>= y 1) (= (2d-vector-ref vec-board (- x 1) (- y 1)) player))
                (and (< x (- width 1)) (< y (- width 1)) (= (2d-vector-ref vec-board (+ x 1) (+ y 1))
player))
                (and (< y (- width 2)) (< x (- width 2)) (= (2d-vector-ref vec-board (+ x 2) (+ y 2))
player))) (set! count (+ count 1))]
              [(and (= (2d-vector-ref vec-board x y) 0)
                (and (> x 1) (>= y 1) (= (2d-vector-ref vec-board (- x 1) (- y 1)) player))
                (and (< x (- width 1)) (< y (- width 1)) (= (2d-vector-ref vec-board (+ x 1) (+ y 1))
player))
                (and (>= y 2) (>= x 2) (= (2d-vector-ref vec-board (- x 2) (- y 2)) player))) (set!
count (+ count 1))]
              [(and (= (2d-vector-ref vec-board x y) 0)
                (and (> x 1) (>= y 1) (= (2d-vector-ref vec-board (- x 1) (- y 1)) player))
                (and (>= y 3) (>= x 3) (= (2d-vector-ref vec-board (- x 2) (- y 2)) player)
                    (= (2d-vector-ref vec-board (- x 3) (- y 3)) player))) (set! count (+ count 1))]
              [(and (= (2d-vector-ref vec-board x y) 0)
                (and (< x (- width 1)) (< y (- width 1)) (= (2d-vector-ref vec-board (+ x 1) (+ y 1))
player))
                (and (< y (- width 3)) (< x (- width 3)) (= (2d-vector-ref vec-board (+ x 2) (+ y 2))
player)
                    (= (2d-vector-ref vec-board (+ x 3) (+ y 3)) player))) (set! count (+ count 1))])))
     count))

  (define (diag2 player)
    (define count 0)
    (define x 0)
    (define y 0)
    (begin
     (for (set! x 2) : (< x 6) : (set! x (+ x 1)) :
       (for (set! y 1) : (< y 6) : (set! y (+ y 1)) :
```

```
        (cond [(and (= (2d-vector-ref vec-board x y) 0)
                    (and (> x 1) (< y (- width 1)) (= (2d-vector-ref vec-board (- x 1) (+ y 1)) player))
                    (and (>= y 1) (< x (- width 1)) (= (2d-vector-ref vec-board (+ x 1) (- y 1)) player))
                    (and (< x (- width 2)) (>= y 2) (= (2d-vector-ref vec-board (+ x 2) (- y 2)) player)))
(set! count (+ count 1))]
              [(and (= (2d-vector-ref vec-board x y) 0)
                    (and (> x 1) (< y (- width 1)) (= (2d-vector-ref vec-board (- x 1) (+ y 1)) player))
                    (and (>= y 1) (< x (- width 1)) (= (2d-vector-ref vec-board (+ x 1) (- y 1)) player))
                    (and (< y (- width 2)) (>= x 2) (= (2d-vector-ref vec-board (- x 2) (+ y 2)) player)))
(set! count (+ count 1))]
              [(and (= (2d-vector-ref vec-board x y) 0)
                    (and (> x 1) (< y (- width 1)) (and (> x 1) (< y (- width 1)) (= (2d-vector-ref
vec-board (- x 1) (+ y 1)) player)))
                    (and (< y (- width 3)) (>= x 3) (= (2d-vector-ref vec-board (- x 2) (+ y 2)) player)
                        (= (2d-vector-ref vec-board (- x 3) (+ y 3)) player))) (set! count (+ count 1))]
              [(and (= (2d-vector-ref vec-board x y) 0)
                    (and (>= y 1) (< x (- width 1)) (= (2d-vector-ref vec-board (+ x 1) (- y 1)) player))
                    (and (< x (- width 3)) (>= y 3) (= (2d-vector-ref vec-board (+ x 2) (- y 2)) player)
                        (= (2d-vector-ref vec-board (+ x 3) (- y 3)) player))) (set! count (+ count 1))]))
    count))
```

New Heuristic Function

```
(define (abstract_four x y player a b)
    (cond  [(and (= (2d-vector-ref vec-board x y) 0)
                 (and (>= y b) (> x a) (< (- x a) width) (< (- y b) width) (= (2d-vector-ref vec-board (- x
a) (- y b)) player))
                 (and (< y (- width b)) (< x (- width a)) (>  (+ x a) 0) (>= (+ y b) 0)  (= (2d-vector-ref
vec-board (+ x a) (+ y b)) player))
                 (and (< y (- width (* 2 b))) (> (+ x (* 2 a)) 0) (>= (+ y (* 2 b)) 0) (< x (- width (* 2 a))) (=
(2d-vector-ref vec-board (+ x (* 2 a)) (+ y (* 2 b))) player))) #t]

          [(and (= (2d-vector-ref vec-board x y) 0)
                 (and (>= y b) (> x a) (< (- x a) width) (< (- y b) width) (= (2d-vector-ref vec-board (- x
a) (- y b)) player))
                 (and (< y (- width b)) (< x (- width a)) (> (+ x a) 0) (>= (+ y b) 0) (= (2d-vector-ref
vec-board (+ x a) (+ y b)) player))
                 (and (>= y (* 2 b)) (> x (* 2 a)) (< (- x (* 2 a)) width) (< (- y (* 2 b)) width) (=
(2d-vector-ref vec-board (- x (* 2 a)) (- y (* 2 b))) player))) #t]

          [(and (= (2d-vector-ref vec-board x y) 0)
                 (and (>= y b) (> x a) (< (- x a) width) (< (- y b) width)  (= (2d-vector-ref vec-board (- x
a) (- y b)) player))
                 (and (>= y (* 2 b)) (> x (* 2 a)) (< (- x (* 2 a)) width) (< (- y (* 2 b)) width)  (=
(2d-vector-ref vec-board (- x (* 2 a)) (- y (* 2 b))) player))
                 (and (>= y (* 3 b)) (> x (* 3 a)) (< (- x (* 3 a)) width) (< (- y (* 3 b)) width)  (=
(2d-vector-ref vec-board (- x (* 3 a)) (- y (* 3 b))) player))) #t]

          [(and (= (2d-vector-ref vec-board x y) 0)
                 (and (< y (- width b)) (< x (- width a)) (> x a) (>= (+ y b) 0) (= (2d-vector-ref vec-board
(+ x a) (+ y b)) player))
                 (and (< y (- width (* 2 b))) (< x (- width (* 2 a))) (> (+ x (* 2 a)) 0) (>= (+ y (* 2 b)) 0) (=
(2d-vector-ref vec-board (+ x (* 2 a)) (+ y (* 2 b))) player))
                 (and (< y (- width (* 3 b))) (< x (- width (* 3 a))) (> (+ x (* 3 a)) 0) (>= (+ y (* 3 b)) 0) (=
(2d-vector-ref vec-board (+ x (* 3 a)) (+ y (* 3 b))) player))) #t]
          [#t #f]))
```

C) Old Copy-2d function Function

```
(define (copy-2d v1 r c)
  (define temp (make-2d-vector r c 0))
  (define (copy-helper i j)
    (cond ((and (< i r) (< j c))
           (begin
             (2d-vector-set! temp i j (2d-vector-ref v1 i j))
             (copy-helper i (+ j 1))))

          ((and (< i r) (= j c))
           (copy-helper (+ i 1) 0))))
  (copy-helper 0 0)
  temp)
```

New Copy-2d Function
(Uses inbuilt vector-copy function)

```
(define (copy-2d v1 r c)
  (define v (make-vector r 0))
  (define (copy-helper i)
    (cond [(< i (- r 1)) (begin
                           (vector-set! v i (vector-copy (vector-ref v1 i))))
                         (copy-helper (+ i 1))]
          [(= i (- r 1)) (vector-set! v i (vector-copy (vector-ref v1 i)))]))
  (begin
    (copy-helper 0)
    v))
```