Mestrado Integrado em Engenharia Informática e Computação

Métodos Formais em Engenharia de Software

Mastermind

Relatório Final

Turma 5 - Grupo I:
201202772 - Henrique Ferrolho - ei12079@fe.up.pt
201202772 - João Pereira - ei12023@fe.up.pt
201208066 - Mário André Ferreira - ei12105@fe.up.pt

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

15 de Dezembro de 2015

Conteúdo

1 Descrição do Sistema

O projeto baseia-se no *Mastermind* que consiste num jogo entre dois jogadores, codificador e descodificador. O codificador tem como objectivo criar uma chave de 4 elementos onde cada um tem 1 de 6 cores diferentes. Já o descodificador tem como objetivo decifrar a chave feita pelo codificador (tendo em conta que o para decifrar a mesma são importantes a cor e a ordem dos elementos). Este tem 10 tentativas para decifrar o código sendo que no final de cada uma o codificador tem de revelar quantas cores estão simultaneamente presentes na tentativa e na chave mas não coinicidem em termos de localização assim como tem de dizer quantas peças estão corretamente colocadas nessa mesma tentativa.
O codificador sai vitorioso se o descodificador não conseguir desvendar a sua chave nas dez tentativas e perde caso o contrário aconteça.
Para além disto é também pretendido a criação de um campeonato, onde são inseridos o número de jogadores e os seus nomes. Os jogos são gerados de forma aleatória e só avançam no campeonato os vencedores de cada jogo. No final é apresentado o vencedor.
Ao longo do relatório chaves criadas por um codificador e por descodificador serão representadas por *codeMaker* e *codeBreaker* respectivamente.

1.1 Lista de requisitos

Para cumprir o objectivo são necessários os seguintes requisitos:

Requisito	Descrição	Restrições
1	Criar chaves (*codeMaker* e *codeBreaker*).	Elementos usados têm de pertencer ao intervalo entre 1 e 6 inclusive. O tamanho da chave é obrigatoriamente 4.
2	verificar cada tentativa de decifrar o codeMaker	$w = (\sum_{i=1}^{6} min(c_i, g_i)) - b$[1]
3	verificar condição de vitória e mostrar e vencedor	O número de tentativas não pode exceder 10; Se o descodificador for o vencedor, o resultado final será (4, 0), ou seja, 4 elementos no sitio correto e 0 cores mal posicionadas.
4	verificar o número de jogos por etapa do campeonato	Tem de ser metade dos jogadores em jogo no torneio.
5	verificar o número de vencedores por etapa do campeonato	Tem de ser o mesmo número que o número de jogos executados nessa etapa.

[1] http://mathworld.wolfram.com/Mastermind.html

2 UML

De seguida serão apresentados o caso de uso de uma jogada e o diagrama de classes.

2.1 Modelo de Caso de Uso

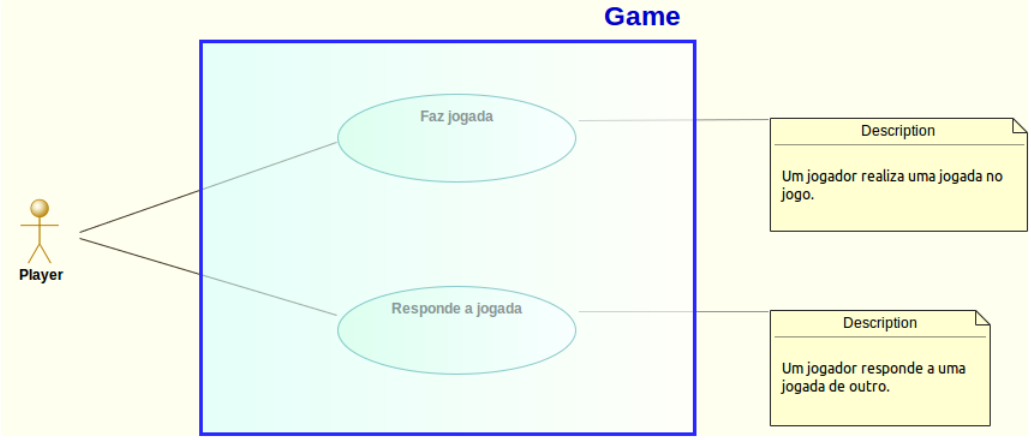

Diagrama de Caso de Uso

Este projecto tem apenas dois casos de uso possíveis, no que diz respeito à interação com o utilizador. Os casos de uso ao a realização de jogadas no jogo.

2.2 Diagrama de Classes

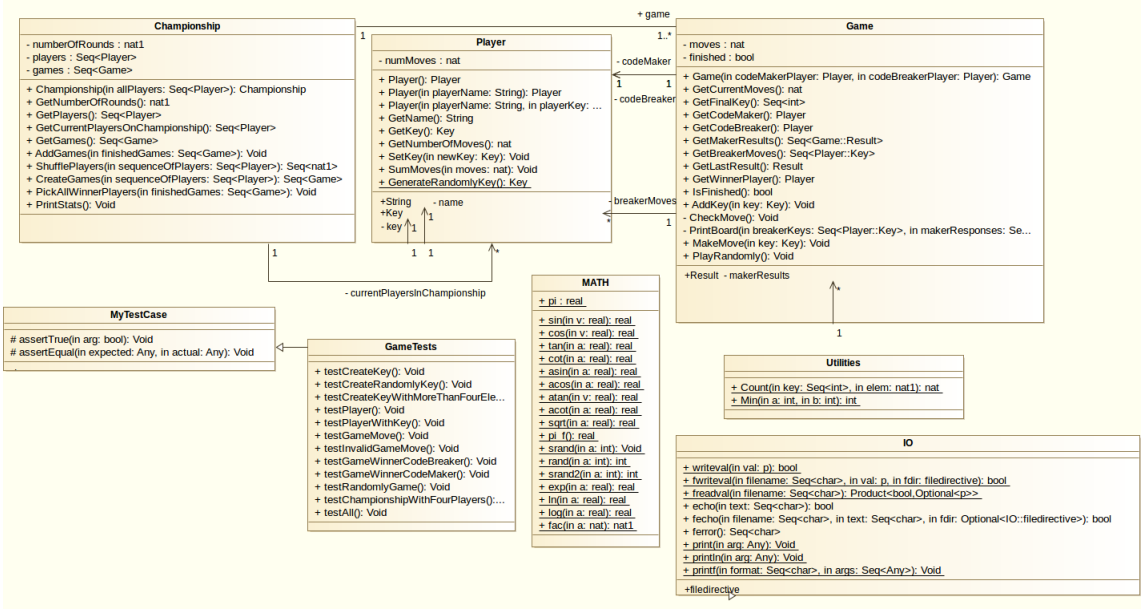

Diagrama de Classes

A classe *Championship* serve de suporte a uma das funcionalidades pedidas na especificação do projeto.

A classe *Player* representa ambos os jogadores existentes no jogo, tanto o *Code Breaker* como o *Code Maker*.

A classe *Game* representa todo o motor de jogo e é nela que estão implementadas as funções mais importantes.

A classe *GameTests* representa os testes unitários do jogo.

A classe *Utilities* serve de apoio a algumas operações gerais usadas em diversas funções.

3 Modelo VDM++

3.1 Championship

```
class Championship -- represents championship

instance variables
 private numberOfRounds : nat1;
 private players : seq of Player := [];
 private currentPlayersInChampionship : seq of Player;
 private games : seq of Game;

 -- only even number of players
 inv len players rem 2 = 0;

operations
 -- creates championship whith the given number of players

 public Championship : seq of Player ==> Championship
  Championship (allPlayers) == (
   numberOfRounds := (MATH`log(len allPlayers) / MATH`log(2));

   players := allPlayers;
   currentPlayersInChampionship := allPlayers;
   games := [];

   return self;
  )
 pre len allPlayers > 1 and len allPlayers rem 2 = 0;

 -- return the number of the championship's rounds

 pure public GetNumberOfRounds : () ==> nat1
  GetNumberOfRounds() == (
   return numberOfRounds;
  );

 -- return all championship's players

 pure public GetPlayers : () ==> seq of Player
  GetPlayers() == (
   return players;
  );


 -- return only the players still in championship

 pure public GetCurrentPlayersOnChampionship : () ==> seq of Player
  GetCurrentPlayersOnChampionship() == (
   return currentPlayersInChampionship;
  );

 -- return all championship's games

 pure public GetGames : () ==> seq of Game
  GetGames() == (
   return games;
  );

 -- add games to championship

 public AddGames : seq of Game ==> ()
  AddGames(finishedGames) == (
   games := games^finishedGames;

   for g in games do (
    g.GetCodeMaker().SumMoves(len g.GetMakerResults());
    g.GetCodeBreaker().SumMoves(len g.GetBreakerMoves());
```

```
   );

  );

  -- shuffle a sequence of players to be picked by CreateGames

  public ShufflePlayers : seq of Player ==> seq of nat1
   ShufflePlayers(sequenceOfPlayers) == (
    dcl playerShuffled : int;
    dcl selectedPlayers : seq of nat1 := [];

    for p in sequenceOfPlayers do (
     if len selectedPlayers = 0
      then
       playerShuffled := MATH`rand(len sequenceOfPlayers) + 1
     else (
      while playerShuffled in set elems selectedPlayers and len selectedPlayers <> len
          sequenceOfPlayers do (
       playerShuffled := MATH`rand(len sequenceOfPlayers) + 1;
      );
     );

     selectedPlayers := selectedPlayers^[playerShuffled];
    );

    return selectedPlayers
  );

  -- pick pairs of two players from the previous shuffled sequence of players

  public CreateGames : seq of Player ==> seq of Game
   CreateGames(sequenceOfPlayers) == (
    dcl playersOrder : seq of nat1 := ShufflePlayers(sequenceOfPlayers);
    dcl generatedGames : seq of Game := [];
    dcl game : Game;

    for i = 1 to len sequenceOfPlayers / 2 by 1 do (
     game := new Game(sequenceOfPlayers(playersOrder((i * 2) - 1)), sequenceOfPlayers(
         playersOrder(i * 2)));
     generatedGames := generatedGames^[game];
    );

    return generatedGames;
  );

  -- pick players of all games in a round of the championship

  public PickAllWinnerPlayers : seq of Game ==> ()
   PickAllWinnerPlayers(finishedGames) == (
    dcl winnerPlayers : seq of Player := [];

    for i = 1 to len finishedGames by 1 do (
     winnerPlayers := winnerPlayers^[finishedGames(i).GetWinnerPlayer()];
    );

    currentPlayersInChampionship := winnerPlayers;
  );

  -- returns the number of moves the winner player made during the entire championship

  public PrintStats : () ==> ()
   PrintStats() == (
    IO`print("\nThe winner of championship was ");
    IO`print((hd GetCurrentPlayersOnChampionship()).GetName());
    IO`print(" and has made ");
    IO`print((hd GetCurrentPlayersOnChampionship()).GetNumberOfMoves());
    IO`print(" moves.\n\n");
  );

end Championship
```

3.2 Game

```
class Game -- represents a game between two players

types
 public Result = seq of nat
 inv r == len r = 2 and r(1) + r(2) <= 4;

instance variables
 private moves : nat := 10;
 private codeMaker: Player;
 private codeBreaker: Player;
 private makerResults : seq of Result := [];
 private breakerMoves: seq of Player`Key := [];
 private finished : bool := false;

inv len makerResults = 10 - moves;
inv finished => len makerResults > 0 and len makerResults < 11;
inv forall i in set inds makerResults \ {len makerResults} & makerResults(i) <> [4, 0];

operations
  -- constructor, initializes the game

  public Game: Player * Player ==> Game
   Game(codeMakerPlayer, codeBreakerPlayer) == (
    codeMaker := codeMakerPlayer;
    codeBreaker := codeBreakerPlayer;
    return self
   );

  -- get the current number of moves remaining

  pure public GetCurrentMoves : () ==> nat
   GetCurrentMoves () == return moves;

  -- get the code maker final key

  pure public GetFinalKey : () ==> seq of int
   GetFinalKey () == return codeMaker.GetKey();

  -- get the code maker player

  pure public GetCodeMaker : () ==> Player
   GetCodeMaker () == return codeMaker;

  -- get the code breaker player

  pure public GetCodeBreaker : () ==> Player
   GetCodeBreaker () == return codeBreaker;

  -- return maker results

  pure public GetMakerResults : () ==> seq of Result
   GetMakerResults () == return makerResults;

  -- return breaker moves

  pure public GetBreakerMoves : () ==> seq of Player`Key
   GetBreakerMoves () == return breakerMoves;

  -- get last result of the maker moves

  pure public GetLastResult : () ==> Result
   GetLastResult() == return makerResults(len makerResults);

  -- return the winner player

  pure public GetWinnerPlayer : () ==> Player
   GetWinnerPlayer() == (
```

```
   if GetLastResult() = [4, 0]
    then
     return codeBreaker
   else
    return codeMaker;
 );

-- tell if a game is finished or not

pure public IsFinished : () ==> bool
 IsFinished() == (
  return finished;
 );

-- adds a new key to the set of the breaker moves and sets the current key of the breaker
    player

public AddKey : Player`Key ==> ()
 AddKey (key) == (
  codeBreaker.SetKey(key);
  breakerMoves := breakerMoves^[key];
 )
pre len key > 0 and len key < 5 and forall x in set elems key & (x > 0 and x < 7)
post len breakerMoves = len breakerMoves~ + 1;

-- checks the move of the breaker player

private CheckMove : () ==> ()
CheckMove () == (
 dcl e_corrects : nat := 0;
 dcl e_exists : nat := 0;
 dcl makerElemRep : nat := 0;
 dcl breakerElemRep : nat := 0;
 dcl sumMins : nat := 0;
 dcl breakerKey : Player`Key := codeBreaker.GetKey();
 dcl makerKey : Player`Key := codeMaker.GetKey();

 for i = 1 to len codeBreaker.GetKey() by 1 do (
   if breakerKey(i) = makerKey(i)
    then
     e_corrects := e_corrects + 1;
 );

 for i = 1 to 6 by 1 do (
  makerElemRep := Utilities`Count(makerKey, i);
  breakerElemRep := Utilities`Count(breakerKey, i);
  sumMins := sumMins + Utilities`Min(makerElemRep, breakerElemRep);
 );

 e_exists := sumMins - e_corrects;

 atomic (
  moves := moves - 1;
  makerResults := makerResults^[[e_corrects, e_exists]];
  finished := e_corrects = 4
 );
);

-- print board

private PrintBoard : seq of Player`Key * seq of Result ==> ()
 PrintBoard (breakerKeys, makerResponses) == (
  dcl currentMove : nat := 10 - moves;

  IO`print("Board Game - Move: ");
  IO`println(currentMove);
  IO`println("MOVE                RESULT [Corrects, Exists]");

  for i = len makerResponses to 1 by -1 do (
   IO`print(breakerKeys(i));
```

```
    IO`print("         ");
    IO`println(makerResponses(i));
   );

   IO`print("Moves Remaing: ");
   IO`println(moves);
   IO`print("\n");
  );

 -- make a move

 public MakeMove : (Player`Key) ==> ()
  MakeMove (key) == (
  if moves > 0
   then (
    if not finished
     then (
      AddKey(key);
      CheckMove();

      PrintBoard(breakerMoves, makerResults);

      if finished
       then (
        IO`print(codeBreaker.GetName());
        IO`println(" won the game.");
        IO`print("The key was ");
        IO`print(codeBreaker.GetKey());
        IO`print(" and the number of tries was ");
        IO`print(len makerResults);
        IO`println(".");
        IO`print("\n");
       )
      elseif moves = 0 and not finished
       then (
        finished := true;
        IO`print(codeMaker.GetName());
        IO`println(" won the game.");
        IO`print("The key was ");
        IO`print(GetFinalKey());
        IO`print(" and the number of moves was ");
        IO`print(len makerResults);
        IO`println(".");
        IO`print("\n");
       );
     )
    else (
     IO`println("The game is over.");
     IO`print("\n");
    )
   )
  else (
   IO`println("The game is over.");
   IO`print("\n");
  )
 )
 pre len key > 0 and len key < 5 and forall x in set elems key & (x > 0 and x < 7);

 -- make a full random game

 public PlayRandomly : () ==> ()
  PlayRandomly() == (
   if moves = 10
    then
     codeMaker.SetKey(Player`GenerateRandomlyKey());

   while not finished do (
    MakeMove(Player`GenerateRandomlyKey());
   );
  );
```

```
end Game
```

3.3 Player

```
class Player -- represents the player of the game

types
 public String = seq of char;
 public Key = seq of nat1
 inv k == len k = 4 and forall x in set elems k & (x > 0 and x < 7);

instance variables
 private name : String := [];
 private key : Key;
 private numMoves : nat := 0;

operations
 -- default constructor of the player class

 public Player : () ==> Player
   Player () == (
    name := "Default";
    return self
   );

  -- constructor of the player class with a string parameter
  public Player : String ==> Player
   Player (playerName) == (
    name := playerName;
    return self
   )
  pre len playerName > 0;

  -- constructor of the player class with a string parameter
  public Player: String * Key ==> Player
   Player (playerName, playerKey) == (
    name := playerName;
    key := playerKey;
    return self
   )
  pre len playerKey = 4 and forall x in set elems playerKey & (x > 0 and x < 7) and len
      playerName > 0;

  -- returns the player's name

  pure public GetName : () ==> String
   GetName () == (
    return name;
   );

  -- returns the player's current key

  pure public GetKey: () ==> Key
   GetKey () == (
    return key;
   );

  -- returns player's the number of moves

  pure public GetNumberOfMoves : () ==> nat
   GetNumberOfMoves () == (
    return numMoves;
   );

  -- sets the player's current key to a new one
```

```
  public SetKey : Key ==> ()
  SetKey (newKey) == (
   key := newKey;
  )
 pre len newKey = 4 and forall x in set elems newKey & (x > 0 and x < 7);

  -- adds a number of moves to the player's current number of moves

  public SumMoves : nat ==> ()
   SumMoves (moves) == (
    numMoves := numMoves + moves;
   );

  -- returns a random key

  public static GenerateRandomlyKey : () ==> Key
  GenerateRandomlyKey() == (
   dcl randomKey : seq of nat1 := [];

   for i = 1 to 4 by 1 do (
    randomKey := randomKey^[MATH`rand(6) + 1];
   );

   return randomKey;
  );

end Player
```

3.4 Utilities

```
class Utilities -- class with some pratical function

operations
 -- returns the number of occurences of a value on a sequence

 public static Count: seq of int * nat1 ==> nat
  Count(key, elem) == (
   dcl count : nat := 0;

   for e in key do
    if e = elem
     then
      count := count + 1;

   return count;
  );

 -- returns the minimum between two values

 public static Min: int * int ==> int
  Min(a, b) == (
   if a < b
    then
     return a;

   return b;
  );

end Utilities
```

4 Validação do Modelo - Testes

Os testes de seguida apresentados foram executados com sucesso na totalidade.

4.1 *GameTests*

```
class GameTests is subclass of MyTestCase -- represents the test suite

operations

 -- check if a key is correctly created

 public testCreateKey : () ==> ()
  testCreateKey() == (
   dcl key : Player`Key := [1, 2, 3, 4];

   assertEqual([1, 2, 3, 4], key);
   assertEqual(4, len key);

   for x in key do (
    assertTrue(x >= 1 and x <= 6);
   )
  );

 -- check if a key is correctly created

 public testCreateRandomlyKey : () ==> ()
  testCreateRandomlyKey() == (
   dcl key : Player`Key := Player`GenerateRandomlyKey();

   assertEqual(4, len key);
   for x in key do (
    assertTrue(x >= 1 and x <= 6);
   )
  );

  -- check if a key can be created with five elements

 public testCreateKeyWithMoreThanFourElems : () ==> ()
  testCreateKeyWithMoreThanFourElems() == (
   dcl key : Player`Key;

   key := [1, 2, 3, 4, 5];
  );

 -- checks if a default player is created with a certain key

 public testPlayer : () ==> ()
  testPlayer() == (
  dcl player: Player := new Player();
  player.SetKey([1, 2, 3, 4]);
  assertEqual([1, 2, 3, 4], player.GetKey());
  assertEqual("Default", player.GetName());
  );

 -- checks if a player is created with a certain key

 public testPlayerWithKey : () ==> ()
  testPlayerWithKey() == (
  dcl player: Player := new Player("Cristo", [1, 2, 3, 4]);
  assertEqual([1, 2, 3, 4], player.GetKey());
  );

 -- check if the sequence returns a correct value

 public testGameMove : () ==> ()
  testGameMove() == (
```

```
  dcl codeMaker: Player := new Player("CodeMaker", [6, 6, 5, 2]);
   dcl codeBreaker: Player := new Player("CodeBreaker");
   dcl g: Game := new Game(codeMaker, codeBreaker);

   -- the result of the game initial moves is 10
   assertEqual(10, g.GetCurrentMoves());

   -- code breaker makes the move [1, 1, 2, 2], and the expected result is (1, 0)
   g.MakeMove([1, 1, 2, 2]);
   assertEqual([1, 0], g.GetLastResult());
 );

-- check if a invalid move is rejected

public testInvalidGameMove : () ==> ()
 testInvalidGameMove() == (
  dcl codeMaker: Player := new Player("CodeMaker", [6, 6, 5, 2]);
   dcl codeBreaker: Player := new Player("CodeBreaker");
   dcl g: Game := new Game(codeMaker, codeBreaker);

   -- the result of the game initial moves is 10
   assertEqual(10, g.GetCurrentMoves());

   -- code breaker makes the move [1, 1, 7, 2], and it should be rejected
   g.MakeMove([1, 1, 7, 2]);
 );

-- simulates a game with the code breaker as the winner

 public testGameWinnerCodeBreaker : () ==> ()
  testGameWinnerCodeBreaker() == (
   -- create game players
   dcl codeMaker: Player := new Player("CodeMaker", [6, 6, 5, 2]);
   dcl codeBreaker: Player := new Player("CodeBreaker");

   -- create game
   dcl g: Game := new Game(codeMaker, codeBreaker);

   -- the result of the game initial moves is 10
   assertEqual(10, g.GetCurrentMoves());

   -- code breaker makes the move [1, 1, 2, 2], and the expected result is (1, 0)
   g.MakeMove([1, 1, 2, 2]);
   assertEqual([1, 0], g.GetLastResult());

   -- code breaker makes the move [3, 3, 4, 4], and the expected result is (0, 0)
   g.MakeMove([3, 3, 4, 4]);
   assertEqual([0, 0], g.GetLastResult());

   -- code breaker makes the move [5, 5, 6, 6], and the expected result is (0, 3)
   g.MakeMove([5, 5, 6, 6]);
   assertEqual([0, 3], g.GetLastResult());

   -- code breaker makes the move [5, 6, 5, 2], and the expected result is (3, 0)
   g.MakeMove([5, 6, 5, 2]);
   assertEqual([3, 0], g.GetLastResult());

   -- code breaker makes the move [6, 6, 5, 2], the expected result is (4, 0) and he wins the
        game
   g.MakeMove([6, 6, 5, 2]);
   assertEqual([4, 0], g.GetLastResult());

   -- the winner was the code breaker
   assertEqual("CodeBreaker", g.GetWinnerPlayer().GetName());

   IO`print("Winner is: ");
   IO`println(g.GetWinnerPlayer().GetName());
  );

  -- simulates a game with the code maker as the winner
```

```
 public testGameWinnerCodeMaker : () ==> ()
 testGameWinnerCodeMaker() == (
  -- create game players
  dcl codeMaker: Player := new Player("CodeMaker", [1, 2, 4, 1]);
  dcl codeBreaker: Player := new Player("CodeBreaker");

  -- create game
  dcl g: Game := new Game(codeMaker, codeBreaker);

  -- the result of the game initial moves is 10
  assertEqual(10, g.GetCurrentMoves());

  -- code breaker makes the move [1, 1, 2, 2], and the expected result is (1, 2)
  g.MakeMove([1, 1, 2, 2]);
  assertEqual([1, 2], g.GetLastResult());

  -- code breaker makes the move [3, 3, 4, 4], and the expected result is (1, 0)
  g.MakeMove([3, 3, 4, 4]);
  assertEqual([1, 0], g.GetLastResult());

  -- code breaker makes the move [5, 5, 6, 6], and the expected result is (0, 0)
  g.MakeMove([5, 5, 6, 6]);
  assertEqual([0, 0], g.GetLastResult());

  -- code breaker makes the move [2, 1, 3, 4], and the expected result is (0, 3)
  g.MakeMove([2, 1, 3, 4]);
  assertEqual([0, 3], g.GetLastResult());

  -- code breaker makes the move [1, 1, 1, 1], and the expected result is (2, 0)
  g.MakeMove([1, 1, 1, 1]);
  assertEqual([2, 0], g.GetLastResult());

  -- code breaker makes the move [2, 2, 2, 2], and the expected result is (1, 0)
  g.MakeMove([2, 2, 2, 2]);
  assertEqual([1, 0], g.GetLastResult());

  -- code breaker makes the move [1, 1, 3, 4], and the expected result is (1, 2)
  g.MakeMove([1, 1, 3, 4]);
  assertEqual([1, 2], g.GetLastResult());

  -- code breaker makes the move [2, 1, 3, 4], and the expected result is (0, 3)
  g.MakeMove([2, 1, 3, 4]);
  assertEqual([0, 3], g.GetLastResult());

  -- code breaker makes the move [1, 1, 2, 4], and the expected result is (1, 3)
  g.MakeMove([1, 1, 2, 4]);
  assertEqual([1, 3], g.GetLastResult());

  -- code breaker makes the move [1, 2, 1, 4], the expected result is (2, 2) and he wins the
       game
  g.MakeMove([1, 2, 1, 4]);
  assertEqual([2, 2], g.GetLastResult());

  -- the winner was the code breaker
  assertEqual("CodeMaker", g.GetWinnerPlayer().GetName());

  IO`print("Winner is: ");
  IO`println(g.GetWinnerPlayer().GetName());
 );

-- simulates a game with randomly moves

public testRandomlyGame : () ==> ()
 testRandomlyGame() == (
  dcl codeMaker: Player := new Player("CodeMaker", [1, 2, 4, 1]);
  dcl codeBreaker: Player := new Player("CodeBreaker");
  dcl g: Game := new Game(codeMaker, codeBreaker);

  -- the result of the game initial moves is 10
```

```
    assertEqual(10, g.GetCurrentMoves());

    -- play the game with randomly moves
    g.PlayRandomly();

    -- the game need to be finished
    assertEqual(true, g.IsFinished());

    IO`print("Winner is: ");
    IO`println(g.GetWinnerPlayer().GetName());
   );

  -- simulates a championship, where four players play randomly

  public testChampionshipWithFourPlayers : () ==> ()
   testChampionshipWithFourPlayers() == (
    -- creating four players to the championship
    dcl p1 : Player := new Player("Tito");
    dcl p2 : Player := new Player("Cristina");
    dcl p3 : Player := new Player("Jacinto");
    dcl p4 : Player := new Player("Ana");
    dcl games : seq of Game;

    -- creating a championship
    dcl champ : Championship := new Championship([p1, p2, p3, p4]);
    assertEqual(4, len champ.GetPlayers());

    -- run championship by rounds
    for i = 1 to champ.GetNumberOfRounds() by 1 do (
     -- creates a championship round with the selected players
     games := champ.CreateGames(champ.GetCurrentPlayersOnChampionship());
     if i = 1
      then (
       assertEqual(2, len games);
      )
     elseif i = 2
      then (
       assertEqual(1, len games);
      );

     for j = 1 to len games by 1 do (
      -- play games randomly
      games(j).PlayRandomly();
     );

     -- get all the winner players of the championship round
     champ.PickAllWinnerPlayers(games);

     if i = 1
      then (
       assertEqual(2, len champ.GetCurrentPlayersOnChampionship());
      )
     elseif i = 2
      then (
       assertEqual(1, len champ.GetCurrentPlayersOnChampionship());
      );

     for j = 1 to len champ.GetCurrentPlayersOnChampionship() by 1 do (
      IO`print("Winner of Game ");
      IO`print(j);
      IO`print(" is: ");
      IO`println(champ.GetCurrentPlayersOnChampionship()(j).GetName());
     );

     -- adding games to the championship history
     champ.AddGames(games);

    );
     -- prints the total moves of the championship winner player
     champ.PrintStats();
```

```
  );

  -- simulates a championship with an odd number of players

  public testChampionshipWithOddNumberOfPlayers : () ==> ()
   testChampionshipWithOddNumberOfPlayers() == (
    -- creating four players to the championship
    dcl p1 : Player := new Player("Tito");
    dcl p2 : Player := new Player("Cristina");
    dcl p3 : Player := new Player("Jacinto");
    dcl p4 : Player := new Player("Ana");
    dcl p5 : Player := new Player("Antonio");

    dcl champ : Championship;

    -- creating a championship
    champ := new Championship([p1, p2, p3, p4, p5]);
   );


  public testAll: () ==> ()
  testAll() == (
   --testCreateKey();
   --testCreateRandomlyKey();
   --testCreateKeyWithMoreThanFourElems();
    --testPlayer();
    --testPlayerWithKey();
    --testGameMove();
    --testInvalidGameMove();
    --testGameWinnerCodeBreaker();
    --testGameWinnerCodeMaker();
    --testRandomlyGame();
    --testChampionshipWithFourPlayers();
    testChampionshipWithOddNumberOfPlayers();
  );

end GameTests
```

4.2 *MyTestCase*

```
class MyTestCase

operations
 -- Simulates assertion checking by reducing it to pre-condition checking.
 -- If 'arg' does not hold, a pre-condition violation will be signaled.

 protected assertTrue: bool ==> ()
 assertTrue(arg) ==
  return
 pre arg;

 -- Simulates assertion checking by reducing it to post-condition checking.
 -- If values are not equal, prints a message in the console and generates
 -- a post-conditions violation.

 protected assertEqual: ? * ? ==> ()
 assertEqual(expected, actual) ==
  if expected <> actual then (
    IO`print("Actual value (");
    IO`print(actual);
    IO`print(") different from expected (");
    IO`print(expected);
    IO`println(")\n")
  )
 post expected = actual;
```

```
end MyTestCase
```

4.3 Resultados

```
261  public testAll: () ==> ()
262  testAll() == (
263    testCreateKey();
264    --testCreateRandomlyKey();
265    --testCreateKeyWithMoreThanFourElems();
266    --testPlayer();
267    --testPlayerWithKey();
268    --testGameMove();
269    --testInvalidGameMove();
270    --testGameWinnerCodeBreaker();
271    --testGameWinnerCodeMaker();
272    --testRandomlyGame();
273    --testChampionshipWithFourPlayers();
274  );
275
276 end GameTests
```

Problems Tasks VDM Quick Interpreter Debug Console

```
<terminated> [Debug Console] testGame [VDM PP Model] Overture debugger
**
** Overture Console
**

new GameTests().testAll() = ()
```

Criação de uma chave válida

```
261  public testAll: () ==> ()
262  testAll() == (
263    --testCreateKey();
264    testCreateRandomlyKey();
265    --testCreateKeyWithMoreThanFourElems();
266    --testPlayer();
267    --testPlayerWithKey();
268    --testGameMove();
269    --testInvalidGameMove();
270    --testGameWinnerCodeBreaker();
271    --testGameWinnerCodeMaker();
272    --testRandomlyGame();
273    --testChampionshipWithFourPlayers();
274  );
275
276end GameTests
```

Problems Tasks VDM Quick Interpreter Debug Console

```
<terminated> [Debug Console] testGame [VDM PP Model] Overture debugger
**
** Overture Console
**

new GameTests().testAll() = ()
```

Criação de uma chave válida aleatória

```
261  public testAll: () ==> ()
262  testAll() == (
263    --testCreateKey();
264    --testCreateRandomlyKey();
265    --testCreateKeyWithMoreThanFourElems();
266    testPlayer();
267    testPlayerWithKey();
268    --testGameMove();
269    --testInvalidGameMove();
270    --testGameWinnerCodeBreaker();
271    --testGameWinnerCodeMaker();
272    --testRandomlyGame();
273    --testChampionshipWithFourPlayers();
274  );
275
276end GameTests
```

Problems Tasks VDM Quick Interpreter Debug Console

```
<terminated> [Debug Console] testGame [VDM PP Model] Overture debugger
**
** Overture Console
**

new GameTests().testAll() = ()
```

Criação de utilizadores

```
261  public testAll: () ==> ()
262  testAll() == (
263    --testCreateKey();
264    --testCreateRandomlyKey();
265    --testCreateKeyWithMoreThanFourElems();
266    --testPlayer();
267    --testPlayerWithKey();
268    testGameMove();
269    --testInvalidGameMove();
270    --testGameWinnerCodeBreaker();
271    --testGameWinnerCodeMaker();
272    --testRandomlyGame();
273    --testChampionshipWithFourPlayers();
274  );
275
276end GameTests
```

Problems Tasks VDM Quick Interpreter Debug Console

```
<terminated> [Debug Console] testGame [VDM PP Model] Overture debugger
**
** Overture Console
**
Board Game - Move: 1
MOVE                RESULT [Corrects, Exists]
[1, 1, 2, 2]        [1, 0]
Moves Remaing: 9


new GameTests().testAll() = ()
```

Teste de uma jogada

```
Board Game - Move: 1
MOVE                RESULT [Corrects, Exists]
[1, 1, 2, 2]        [1, 2]
Moves Remaing: 9

Board Game - Move: 2
MOVE                RESULT [Corrects, Exists]
[3, 3, 4, 4]        [1, 0]
[1, 1, 2, 2]        [1, 2]
Moves Remaing: 8

Board Game - Move: 3
MOVE                RESULT [Corrects, Exists]
[5, 5, 6, 6]        [0, 0]
[3, 3, 4, 4]        [1, 0]
[1, 1, 2, 2]        [1, 2]
Moves Remaing: 7

Board Game - Move: 4
MOVE                RESULT [Corrects, Exists]
[2, 1, 3, 4]        [0, 3]
[5, 5, 6, 6]        [0, 0]
[3, 3, 4, 4]        [1, 0]
[1, 1, 2, 2]        [1, 2]
Moves Remaing: 6

Board Game - Move: 5
MOVE                RESULT [Corrects, Exists]
[1, 1, 1, 1]        [2, 0]
[2, 1, 3, 4]        [0, 3]
[5, 5, 6, 6]        [0, 0]
[3, 3, 4, 4]        [1, 0]
[1, 1, 2, 2]        [1, 2]
Moves Remaing: 5

Board Game - Move: 6
MOVE                RESULT [Corrects, Exists]
[2, 2, 2, 2]        [1, 0]
[1, 1, 1, 1]        [2, 0]
[2, 1, 3, 4]        [0, 3]
[5, 5, 6, 6]        [0, 0]
[3, 3, 4, 4]        [1, 0]
[1, 1, 2, 2]        [1, 2]
Moves Remaing: 4

Board Game - Move: 7
MOVE                RESULT [Corrects, Exists]
[1, 1, 3, 4]        [1, 2]
[2, 2, 2, 2]        [1, 0]
[1, 1, 1, 1]        [2, 0]
[2, 1, 3, 4]        [0, 3]
[5, 5, 6, 6]        [0, 0]
[3, 3, 4, 4]        [1, 0]
[1, 1, 2, 2]        [1, 2]
Moves Remaing: 3

Board Game - Move: 8
MOVE                RESULT [Corrects, Exists]
[2, 1, 3, 4]        [0, 3]
[1, 1, 3, 4]        [1, 2]
[2, 2, 2, 2]        [1, 0]
[1, 1, 1, 1]        [2, 0]
[2, 1, 3, 4]        [0, 3]
[5, 5, 6, 6]        [0, 0]
[3, 3, 4, 4]        [1, 0]
[1, 1, 2, 2]        [1, 2]
Moves Remaing: 2

Board Game - Move: 9
MOVE                RESULT [Corrects, Exists]
[1, 1, 2, 4]        [1, 3]
[2, 1, 3, 4]        [0, 3]
[1, 1, 3, 4]        [1, 2]
[2, 2, 2, 2]        [1, 0]
[1, 1, 1, 1]        [2, 0]
[2, 1, 3, 4]        [0, 3]
[5, 5, 6, 6]        [0, 0]
[3, 3, 4, 4]        [1, 0]
[1, 1, 2, 2]        [1, 2]
Moves Remaing: 1

Board Game - Move: 10
MOVE                RESULT [Corrects, Exists]
[1, 2, 1, 4]        [2, 2]
[1, 1, 2, 4]        [1, 3]
[2, 1, 3, 4]        [0, 3]
[1, 1, 3, 4]        [1, 2]
[2, 2, 2, 2]        [1, 0]
[1, 1, 1, 1]        [2, 0]
[2, 1, 3, 4]        [0, 3]
[5, 5, 6, 6]        [0, 0]
[3, 3, 4, 4]        [1, 0]
[1, 1, 2, 2]        [1, 2]
Moves Remaing: 0

CodeMaker won the game.
The key was [1, 2, 4, 1] and the number of moves was 10.

Winner is: CodeMaker
```

Teste da vitória do codificador

```
Board Game - Move: 1
MOVE                  RESULT [Corrects, Exists]
[1, 1, 2, 2]          [1, 0]
Moves Remaing: 9

Board Game - Move: 2
MOVE                  RESULT [Corrects, Exists]
[3, 3, 4, 4]          [0, 0]
[1, 1, 2, 2]          [1, 0]
Moves Remaing: 8

Board Game - Move: 3
MOVE                  RESULT [Corrects, Exists]
[5, 5, 6, 6]          [0, 3]
[3, 3, 4, 4]          [0, 0]
[1, 1, 2, 2]          [1, 0]
Moves Remaing: 7

Board Game - Move: 4
MOVE                  RESULT [Corrects, Exists]
[5, 6, 5, 2]          [3, 0]
[5, 5, 6, 6]          [0, 3]
[3, 3, 4, 4]          [0, 0]
[1, 1, 2, 2]          [1, 0]
Moves Remaing: 6

Board Game - Move: 5
MOVE                  RESULT [Corrects, Exists]
[6, 6, 5, 2]          [4, 0]
[5, 6, 5, 2]          [3, 0]
[5, 5, 6, 6]          [0, 3]
[3, 3, 4, 4]          [0, 0]
[1, 1, 2, 2]          [1, 0]
Moves Remaing: 5

CodeBreaker won the game.
The key was [6, 6, 5, 2] and the number of tries was 5.

Winner is: CodeBreaker
```

Teste da vitória do descodificador

```
Winner of Game 1 is: Tito
Winner of Game 2 is: Ana
Board Game - Move: 1
MOVE                RESULT [Corrects, Exists]
[2, 5, 3, 5]        [1, 0]
Moves Remaing: 9

Board Game - Move: 2
MOVE                RESULT [Corrects, Exists]
[5, 3, 5, 2]        [0, 1]
[2, 5, 3, 5]        [1, 0]
Moves Remaing: 8

Board Game - Move: 3
MOVE                RESULT [Corrects, Exists]
[6, 2, 4, 3]        [0, 2]
[5, 3, 5, 2]        [0, 1]
[2, 5, 3, 5]        [1, 0]
Moves Remaing: 7

Board Game - Move: 4
MOVE                RESULT [Corrects, Exists]
[4, 2, 2, 5]        [0, 1]
[6, 2, 4, 3]        [0, 2]
[5, 3, 5, 2]        [0, 1]
[2, 5, 3, 5]        [1, 0]
Moves Remaing: 6

Board Game - Move: 5
MOVE                RESULT [Corrects, Exists]
[6, 2, 2, 6]        [1, 1]
[4, 2, 2, 5]        [0, 1]
[6, 2, 4, 3]        [0, 2]
[5, 3, 5, 2]        [0, 1]
[2, 5, 3, 5]        [1, 0]
Moves Remaing: 5

Board Game - Move: 6
MOVE                RESULT [Corrects, Exists]
[3, 4, 2, 6]        [1, 1]
[6, 2, 2, 6]        [1, 1]
[4, 2, 2, 5]        [0, 1]
[6, 2, 4, 3]        [0, 2]
[5, 3, 5, 2]        [0, 1]
[2, 5, 3, 5]        [1, 0]
Moves Remaing: 4

Board Game - Move: 7
MOVE                RESULT [Corrects, Exists]
[4, 3, 6, 1]        [0, 2]
[3, 4, 2, 6]        [1, 1]
[6, 2, 2, 6]        [1, 1]
[4, 2, 2, 5]        [0, 1]
[6, 2, 4, 3]        [0, 2]
[5, 3, 5, 2]        [0, 1]
[2, 5, 3, 5]        [1, 0]
Moves Remaing: 3

Board Game - Move: 8
MOVE                RESULT [Corrects, Exists]
[4, 3, 3, 1]        [0, 1]
[4, 3, 6, 1]        [0, 2]
[3, 4, 2, 6]        [1, 1]
[6, 2, 2, 6]        [1, 1]
[4, 2, 2, 5]        [0, 1]
[6, 2, 4, 3]        [0, 2]
[5, 3, 5, 2]        [0, 1]
[2, 5, 3, 5]        [1, 0]
Moves Remaing: 2

Board Game - Move: 9
MOVE                RESULT [Corrects, Exists]
[1, 2, 5, 3]        [0, 2]
[4, 3, 3, 1]        [0, 1]
[4, 3, 6, 1]        [0, 2]
[3, 4, 2, 6]        [1, 1]
[6, 2, 2, 6]        [1, 1]
[4, 2, 2, 5]        [0, 1]
[6, 2, 4, 3]        [0, 2]
[5, 3, 5, 2]        [0, 1]
[2, 5, 3, 5]        [1, 0]
Moves Remaing: 1

Board Game - Move: 10
MOVE                RESULT [Corrects, Exists]
[3, 6, 4, 3]        [0, 1]
[1, 2, 5, 3]        [0, 2]
[4, 3, 3, 1]        [0, 1]
[4, 3, 6, 1]        [0, 2]
[3, 4, 2, 6]        [1, 1]
[6, 2, 2, 6]        [1, 1]
[4, 2, 2, 5]        [0, 1]
[6, 2, 4, 3]        [0, 2]
[5, 3, 5, 2]        [0, 1]
[2, 5, 3, 5]        [1, 0]
Moves Remaing: 0

Ana won the game.
The key was [2, 1, 1, 6] and the number of moves was 10.

Winner of Game 1 is: Ana

The winner of championship was Ana and has made 30 moves.


new GameTests().testAll() = ()
```

Teste do Campeonato

5 Verificação do Modelo - Análise de Consistência

Para analisar a consistência do programa, isto é, verificar que as *pre*, *post* e *invariants* condições são realmente compridas foram feitos testes que tentam violar as mesmas e foi obtido insucesso nos mesmos testes o que significa que o programa é realmente consistente.
Abaixo podem-se encontrar algumas imagens que refletem estes mesmos testes.

Tentativa falhada de criar uma chave com mais de 4 elementos

Tentativa falhada fazer uma jogada inválida

```
public testAll: () ==> ()
testAll() == (
  --testCreateKey();
  --testCreateRandomlyKey();
  --testCreateKeyWithMoreThanFourElems();
  --testPlayer();
  --testPlayerWithKey();
  --testGameMove();
  --testInvalidGameMove();
  --testGameWinnerCodeBreaker();
  --testGameWinnerCodeMaker();
  --testRandomlyGame();
  --testChampionshipWithFourPlayers();
  testChampionshipWithOddNumberOfPlayers();
);

end GameTests
```

```
Problems  Tasks  Console  VDM Quick Interpreter  Debug
[Debug Console] New_configuration [VDM PP Model] Overture debugger
**
** Overture Console
**
Error 4071: Precondition failure: pre_Championship in 'Championship' (/home/joao/git/feup-mfes/mastermind/Championship.vdmpp) at line 24:25
```

Tentativa falhada de criar um campeonato com um número ímpar de jogadores

6 Transformação do Código para Java

A transformação do código de VDM++ para Java foi efetuada com sucesso, sendo que apenas foram precisas duas pequenas correções (2 *casts* de *int* para *long*) descritas abaixo.

Championship.java (linha 105)

```
long toVar_1 = Utils.divide((1.0 * sequenceOfPlayers.size()), 2L);
long toVar_1 = (long) Utils.divide((1.0 * sequenceOfPlayers.size()), 2L);
```

Game.java (linha 130)

```
for (Long i = makerResponses.size();
for (Long i = (long) makerResponses.size();
```

6.1 Mastermind

O grupo decidiu colocar aqui a interface criada para executar o jogo em JAVA.

```
package mastermind;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;

import org.overture.codegen.runtime.VDMSeq;

public class Mastermind {

        public static boolean validOption = false;
        public static BufferedReader br = new BufferedReader(new InputStreamReader(Syste

        public static void main(String[] args) throws IOException {
                startMenu();
        }

        public static void startMenu() throws IOException {
                System.out.println("########################################");
                System.out.println("########################################");
                System.out.println("#######                          #######");
                System.out.println("#######                          #######");
                System.out.println("#######      MasterMind          #######");
                System.out.println("#######                          #######");
                System.out.println("#######                          #######");
                System.out.println("########################################");
                System.out.println("########################################\n");

                System.out.println("\n#Options");
                System.out.println("  1. Play a game");
                System.out.println("  2. Build a Championship");
                System.out.println("  3. Help");
                System.out.println("  4. Credits");

                int option = insertOption();

                switch (option) {
                case 1:
```

```
                PlayAGameOption();
                break;

        case 2:
                BuildAChampionshipOption();
                break;

        case 3:
                ShowHelp();
                break;

        case 4:
                ShowCredits();
                break;

        default:
                System.err.println("Invalid option! Exiting...");
                break;
        }
}

public static int insertOption() throws IOException {
        int option = 0;

        System.out.print("Enter option: ");

        try {
                option = Integer.parseInt(br.readLine());
        } catch (NumberFormatException ex) {
        }

        return option;
}

public static void PlayAGameOption() throws IOException {
        System.out.println("\n#####################################");
        System.out.println("#####################################");
        System.out.println("#######                       #######");
        System.out.println("#######      MasterMind       #######");
        System.out.println("#######                       #######");
        System.out.println("#######   Play a Game Menu    #######");
        System.out.println("#######                       #######");
        System.out.println("#####################################");
        System.out.println("#####################################\n");

        System.out.println("\n#Options");
        System.out.println("  1. vs COM");
        System.out.println("  2. vs Player");
        System.out.println("  3. Back");

        int option = insertOption();

        switch (option) {
        case 1:
                playGameAgainstComputer();
                break;
```

```
            case 2:
                    playGameAgainstPlayer();
                    break;

            case 3:
                    System.out.print("\n");
                    startMenu();
                    break;

            default:
                    System.err.println("Invalid option! Exiting...");
                    break;
            }

    }

    @SuppressWarnings("resource")
    private static void playGameAgainstComputer() throws IOException {
            String name = new String();

            System.out.print("\n#Insert your name: ");
            name = br.readLine();

            System.out.print("\n#Hello " + name + "! Let's start the game :) !\n\n")

            Player codeMaker = new Player("Computer", Player.GenerateRandomlyKey());
            Player codeBreaker = new Player(name);

            Game game = new Game(codeMaker, codeBreaker);

            runGame(game);

            System.out.print("\n\n");

            System.out.println("Press Enter To Return To Start Menu...");
    new Scanner(System.in).nextLine();

            startMenu();
    }

    @SuppressWarnings("resource")
    private static void playGameAgainstPlayer() throws IOException {
            String makerName = new String(), breakerName = new String();
            VDMSeq makerCode = new VDMSeq();

            System.out.print("\n#Code Maker, insert your name: ");
            makerName = br.readLine();

            System.out.print("\n#Hello " + makerName + "! Now insert your code (exam
            makerCode = convertStringToKey(br.readLine());

            System.out.print("\n#Code Breaker, insert your name: ");
            breakerName = br.readLine();

            System.out.print("\n#Hello " + breakerName + "! Let's start the game :)
```

```
        Player codeMaker = new Player(makerName, makerCode);
        Player codeBreaker = new Player(breakerName);

        Game game = new Game(codeMaker, codeBreaker);

        runGame(game);

        System.out.print("\n\n");

        System.out.println("Press Enter To Return To Start Menu...");
new Scanner(System.in).nextLine();

        startMenu();
}

@SuppressWarnings("unchecked")
private static void BuildAChampionshipOption() throws NumberFormatException, IOE
        VDMSeq players = new VDMSeq();

        System.out.print("\n#Insert an even number of players: ");
        int numberOfPlayers= Integer.parseInt(br.readLine());

        for(int i = 0; i < numberOfPlayers; i++) {
                System.out.print("\n#Player " + (i + 1) +", insert your name: ")
                players.add(new Player(br.readLine()));
        }

        Championship c = new Championship(players);

        for(int i = 0; i < c.GetNumberOfRounds().intValue(); i++) {
                VDMSeq games = c.CreateGames(c.GetCurrentPlayersOnChampionship()

                for(int j = 0; j < games.size(); j++) {
                        Game g = (Game)games.get(j);
                        System.out.println("\nGame " + (j + 1) + " - " + g.GetCo
                        playChampionshipGame((Game)games.get(j));
                }

                c.PickAllWinnerPlayers(games);
                c.AddGames(games);
        }

        c.PrintStats();
}

private static void playChampionshipGame(Game game) throws IOException {
        System.out.print("\n#" + game.GetCodeMaker().GetName() + ", you are the

        game.GetCodeMaker().SetKey(convertStringToKey(br.readLine()));

        System.out.print("\n#Let the game begins !\n\n");

        runGame(game);
}
```

```
        private static void runGame(Game game) throws IOException {
                String str = new String();

                while (!game.IsFinished()) {
                        System.out.print("#" + game.GetCodeBreaker().GetName() + ", inse
                        str = br.readLine();

                        game.MakeMove(convertStringToKey(str));
                }
        }

        @SuppressWarnings("unchecked")
        private static VDMSeq convertStringToKey(String str) {
                VDMSeq key = new VDMSeq();
                String[] elems = str.split(",");

                for (int i = 0; i < elems.length; i++)
                        key.add(Long.parseLong(elems[i]));

                return key;
        }

        private static void ShowHelp() {

        }

        @SuppressWarnings("resource")
        private static void ShowCredits() throws IOException {
                System.out.println("\nThis project was developed by ");
                System.out.println("\tHenrique Ferrolho");
                System.out.println("\tJoao Pereira");
                System.out.println("\tMario Macedo");
                System.out.println("using VDM++ on Overture IDE and later generated to J

                System.out.println("Press Enter To Return To Start Menu...");
        new Scanner(System.in).nextLine();

        startMenu();
        }
}
```

7 Conclusão

Face aos requisitos a que nos propusemos, podemos concluir que todos foram concretizados com sucesso, não obstante que haveria sempre espaço para algumas melhorias, como por exemplo, ser o jogador (codificador) a dizer o resultado da tentativa do descodificador ao invés de isto ser feito pelo computador e posteriormente validar o resultado dado por esse mesmo jogador. Outra das possíveis melhorias seria mostras as estatísticas de todos os jogadores no campeonato ordenadas através de um algoritmo de quicksort, cuja tentativa teve insucesso.

Quanto à participação dos elementos do grupo no trabalho destaca-se uma maior participação pela parte do João, razão pela qual é lhe atribuída 40% da participação no projeto. Os restantes 60% são distribuídos equitativamente pelo Henrique e pelo Mário.

8 Referências

Manual de Utilização do VDM-10

Vienna Development Method

Documentação do Overture

Descrição do tema e estrutura do relatório

Descrição e Regras do Mastermind

Exemplo Vending Machine