UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

INSTITUTO METRÓPOLE DIGITAL

TECNOLOGIA DA INFORMAÇÃO

IMD1012 - TURMA T04

INTRODUÇÃO ÀS TÉCNICAS DE PROGRAMAÇÃO

PROFESSOR: ANTONINO FEITOSA

PROJETO FINAL – UNIDADE 3 – 2021.2

DIOGO DA SILVA LIMA - 20210049129

JOÃO PAULO PEREIRA DE MEDEIROS - 20210053293

---

1. Introdução

O game proposto apresenta regras para pesca, sem destruir o meio ambiente, definidas pelo Órgão de Controle do Meio Ambiente (**OCMA**).

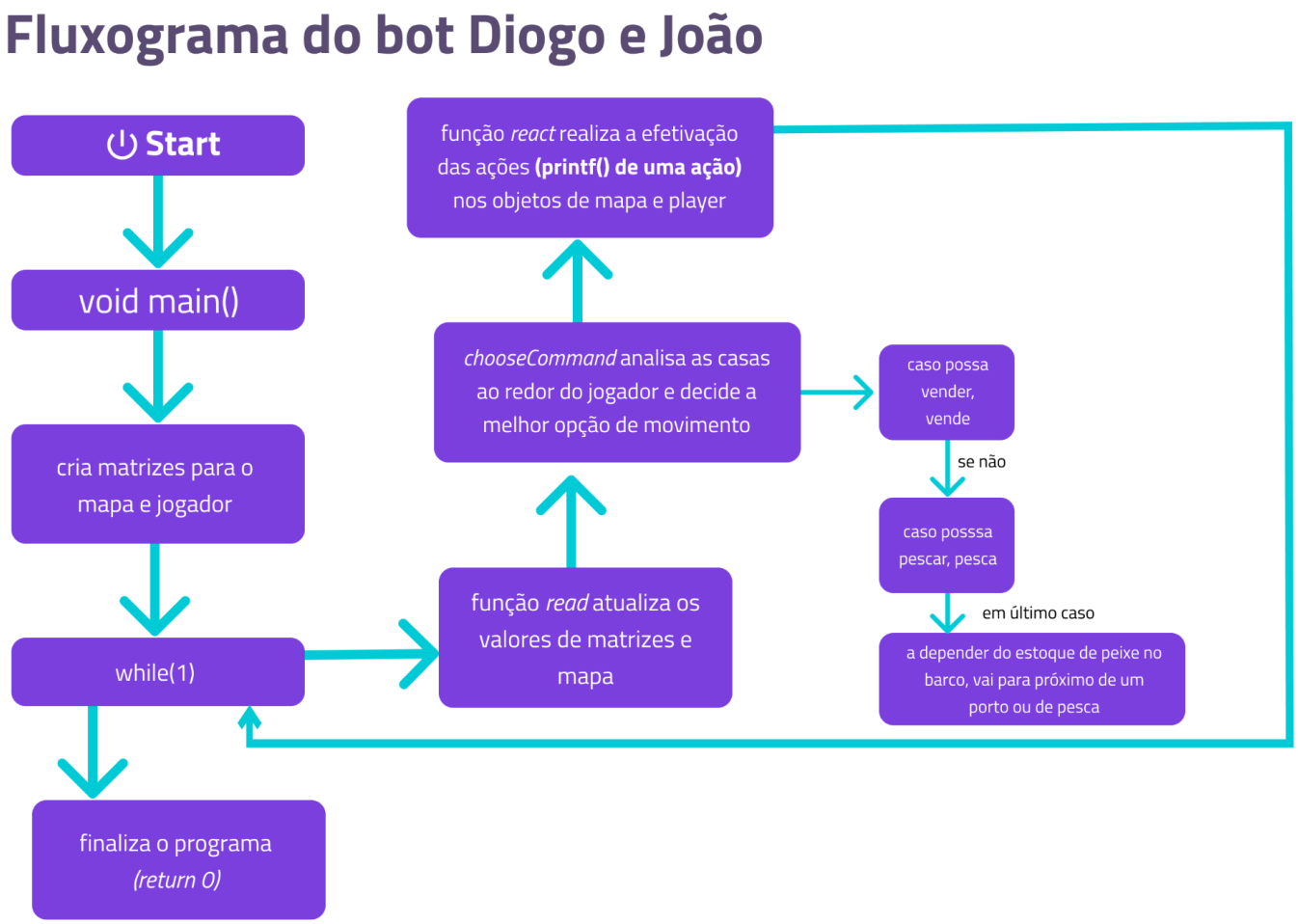
Durante as partidas, bots desenvolvidos previamente com a linguagem de programação C e compilados em seus respectivos arquivos binários (*file.exe*), competem entre si para tentar chegar na marca de R$ 10.000 primeiro, sendo o vencedor que atinge primeiro essa quantia.

Nesse sentido, o objetivo do jogo é realizar pesca de venda de peixes no mar, de modo a não prejudicar o meio ambiente, não sendo, assim, multado pelo órgão de fiscalização ambiente supramencionado (OCMA).

Para a componente curricular Introdução às Técnicas de Programação (IMD1012), o objetivo se mostra a desafiar os alunos para testarem seus conhecimentos consolidados durante a disciplina, a fim de desenvolver bots que competiram entre si.

2. Implementação

versão em **pdf** para ser visualizado melhor: Frame 1.pdf



Anexo (Código-Fonte)

Makefile:

```
CC := gcc

NODE := node


build: message bot1 bot2 bot3 bot4 ocmaRUN
```

```
message:
    @echo "Hello, we're compiling OCMA program file bots"

bot1: src/bot_diogo_joaopaulo.c
    $(CC) $^ -o $@

bot2: src/bot_diogo_joaopaulo1.c
    $(CC) $^ -o $@

bot3: src/bot_diogo_joaopaulO2.c
    $(CC) $^ -o $@

bot4: src/bot_diogo_joaopaulo3.c
    $(CC) $^ -o $@

ocmaRUN: src/ocma.js
    $(NODE) $^ bot1 bot2  bot3 bot4

clean: bot1 bot2 bot3 bot4
    @rm $^

reload: build
```

Nesse sentido, apenas o comando ***make build*** é necessário ser executado no terminal para que a compilação e execução, com o *ocma*, seja feita. O comando ***make clean*** fica a critério, utilizado para remover os antigos

executáveis e executar uma possível nova execução e o ***make reload*** utilizado para executar novamente a ação de build.

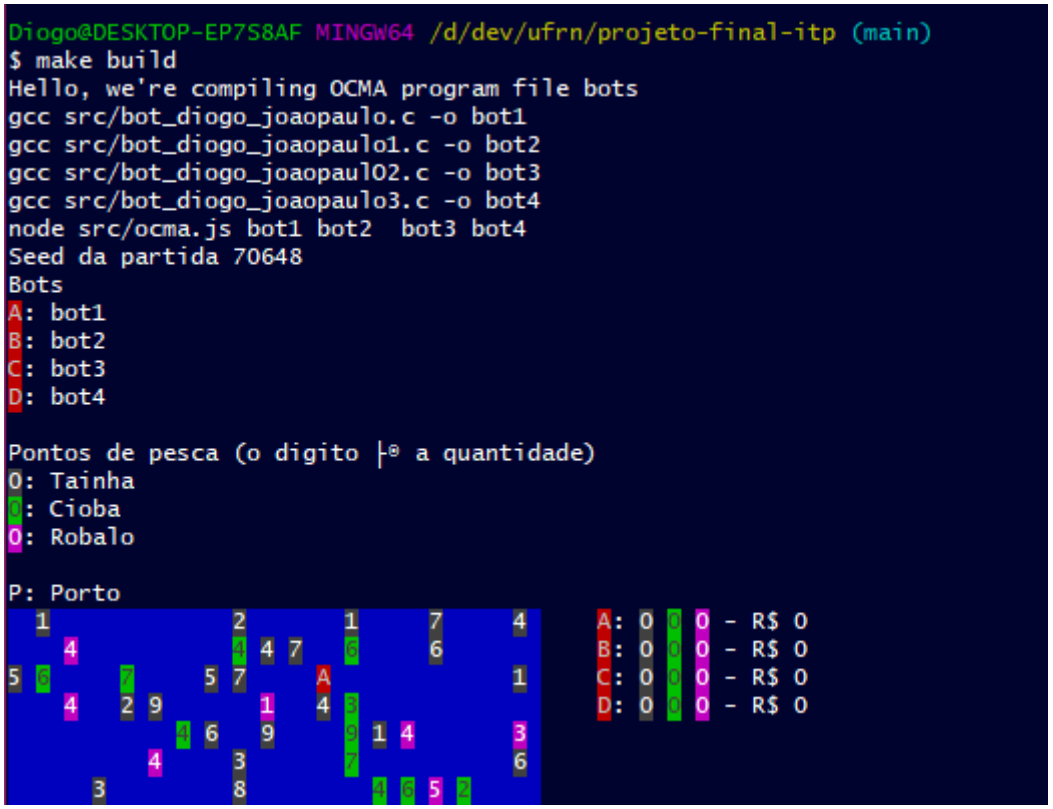


Imagem mostrando a execução utilizando o Makefile.

Código-fonte:

```
/*****************************************************************************
**********

Autores: Diogo Lima e João Paulo M.

Versão:  1.0.0

Disciplina: ITP

Objetivo: Desenvolver um bot escrito em C capaz de ler entradas de dados,
mapeá-las

e, em seguida, utilizá-las para traçar estratégias para ganhar o jogo
ambiental.
```

```
Isto é, identificar pontos de pesca e venda de peixes (portos) mais próximos, mover-se
e ganhar dinheiro até o valor máximo de R$ 10.000,00.
**********/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>

#define MAX_LINE 50

/**
 * @brief ponto do mapa
 *
 */
typedef struct {
  bool anyOtherPlayerOnSurface; //há algum bot na superfície.
   int value; //valor encontrado no ponto. Informa se há peixe ou não, se é local de pesca etc.
} Point;

/**
 * @brief mapa do jogo
 *
 */
typedef struct {
```

```
  int height; //altura do mapa
  int width; //largura do mapa
  int totalBotsPlaying; //quantidade de bots jogando
  Point** points; //pontos de que o mapa é composto
} Map;


/**
 * @brief estoque de peixes do bot
 *
 */
typedef struct {
  int total;
} Stock;


/**
 * @brief player object data structure
 *
 */
typedef struct {
  char id[MAX_LINE]; //id do meu jogador ou bot
  int row; // posição da linha, refere-se ao movimento lateral
  int column; // posição da coluna, refere-se ao movimento vertical
  Stock stock; //estoque de peixes
} Player;



/**
 * @brief position enumeration
```

```
 *
 */
enum Position {
  up,
  down,
  right,
  left,
};


/**
 * @brief informa se ponto é apenas mar aberto
 *
 * @param value
 * @return true
 * @return false
 */
bool isEmptySeaArea(int value) {
  if(value == 0) {
    return true;
  } else {
    return false;
  }
}


/**
 * @brief informa se ponto é um porto
 *
 * @param value
```

```
 * @return true
 * @return false
 */
bool isHarborArea(int value) {
  if(value == 1) {
    return true;
  } else {
    return false;
  }
}

/**
 * @brief informa se região é propícia para pesca (há mais de 1 peixe)
 *
 * @param value
 * @return true
 * @return false
 */
bool isFishingArea(int value) {
   if((value >= 12 && value <= 19) || (value >= 22 && value <= 29) || (value
>= 32 && value <= 39)) {
    return true;
  } else {
    return false;
  }
}

/**
```

```
 * @brief informa se estoque está vazio
 *
 * @param stock
 * @return true
 * @return false
 */
bool isEmptyStock(Stock stock) {
  if(stock.total <= 0) {
    return true;
  } else {
    return false;
  }
}


/**
 * @brief informa se estoque está cheio
 *
 * @param stock
 * @return true
 * @return false
 */
bool isFullStock(Stock stock) {
  if(stock.total >= 10) {
    return true;
  } else {
    return false;
  }
}
```

```c
/**
 * @brief Set the Zero Items On Stock object
 *
 * @param stock
 * @return * limpar
 */
void setZeroItemsOnStock(Stock* stock) {
  stock->total = 0;
}

/**
 * @brief aumenta uma unidade no estoque
 *
 * @param stock
 */
void addItemToStock(Stock* stock) {
  stock->total += 1;
}

/**
  * @brief Calcula distância entre dois pontos do mapa, dadas as suas coordenadas
 *
 * @param x1
 * @param x2
 * @param y1
 * @param y2
```

```
 * @return double
 */
double calculateDistance(int x1, int x2, int y1, int y2) {
  return sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));
}


/**
 * @brief Get the The Nearest Harbor Area object
 *
 * @param player
 * @param map
 * @return int*
 */
int* getTheNearestHarborArea(Player player, Map map) {
  int* coords = calloc(2, sizeof(int));
  double minDistance, distance;


   minDistance = calculateDistance(player.row, (map.height-1), player.column,
(map.width-1));


  for (int i = 0; i < map.height; i++) {
    for (int j = 0; j < map.width; j++) {
      if (map.points[i][j].value == 1) {
        distance = calculateDistance(player.row, i, player.column, j);
        if (distance < minDistance) {
          minDistance = distance;
          coords[0] = i;
          coords[1] = j;
```

```
        }
      }
    }
  }

  return coords;
}

/**
 * @brief retorna coordenadas da área de pesca mais próxima ao bot
 *
 * @param player
 * @param map
 * @return int*
 */
int* getTHeNearestFishingArea(Player player, Map map) {
  int* coords = calloc(2, sizeof(int));
  double minDistance, distance;

  minDistance = calculateDistance(player.row, (map.height-1), player.column,
(map.width-1));

  for (int i = 0; i < map.height; i++) {
    for (int j = 0; j < map.width; j++) {
      if (isFishingArea(map.points[i][j].value)) {
        distance = calculateDistance(player.row, i, player.column, j);
        if (distance < minDistance) {
          minDistance = distance;
```

```
          coords[0] = i;
          coords[1] = j;
        }
      }
    }
  }

  return coords;
}

/**
 * @brief movimenta bot em direção à coordenada
 *
 * @param player
 * @param coords
 * @return char*
 */
char* goTo(Player player, int* coords) {
  char* command = (char*) calloc(MAX_LINE, sizeof(char));

  if (player.column > coords[1]) {
    strcpy(command, "LEFT");
  }
  else if (coords[1] > player.column) {
    strcpy(command, "RIGHT");
  }
  else if (player.column == coords[1]) {
    if (player.row > coords[0]) {
```

```
      strcpy(command, "UP");
    }
    else if (coords[0] > player.row) {
      strcpy(command, "DOWN");
    }
    else if (coords[0] == player.row) {
      strcpy(command, "SELL");
    }
  }

  return command;
}

/**
 * @brief bot se move em direção ao porto, caso estoque esteja cheio. caso contrário, irá se mover em direção à área de pesca mais próxima.
 *
 * @param player
 * @param map
 * @return char*
 */
char* move(Player player, Map map) {
  if(isFullStock(player.stock)) {
    return goTo(player, getTheNearestHarborArea(player, map));
  } else {
    return goTo(player, getTHeNearestFishingArea(player, map));
  }
}
```

```c
/**
 * @brief escolhe comando do bot (movimentação, pesca ou venda) conforme situação do mapa
 *
 * @param player
 * @param map
 * @return char*
 */
char* chooseCommand(Player player, Map map) {
  //valor encontrado na posição atual do bot
  int value = map.points[player.row][player.column].value;

  char* command = (char*) calloc(MAX_LINE, sizeof(char));

  if(isHarborArea(value) && isFullStock(player.stock)) {
    strcpy(command, "SELL");
  } else if(isFishingArea(value) && !isFullStock(player.stock)) {
    strcpy(command, "FISH");
  } else {
    strcpy(command, move(player, map));
  }

  return command;
}

/**
```

```
 * @brief lê os dados do jogo e atualiza o mapa conforme posições dos bots adversários
 *
 * @param player
 * @param map
 */
void read(Player* player, Map* map) {
  map->points = malloc(map->height * sizeof(Point*));

  for (int i = 0; i < map->height; i++){
    map->points[i] = malloc(map->width * sizeof(Point));
  }

  for (int i = 0; i < map->height; i++) {
    for (int j = 0; j < map->width; j++) {
      //seta valor presente em ponto
      scanf("%i", &map->points[i][j].value);
    }
  }

  char id[MAX_LINE];

  int v, x, y;

  //seta quantidade total de bots
  scanf(" BOTS %i", &map->totalBotsPlaying);

  for (int i = 0; i < map->totalBotsPlaying; i++) {
```

```
    scanf("%s %i %i", id, &x, &y);


    //se id for do meu bot, atualizar minha posição
    if(strcmp(player->id, id) == 0) {
      player->row = x;
      player->column = y;
    };


    //seta que há ao menos um bot na superfície do ponto
    map->points[x][y].anyOtherPlayerOnSurface = true;
  }
}


/**
 * @brief reage ao resultado de comandos do bot
 *
 * @param player
 * @param command
 * @param result
 */
void react(Player* player, char* command, char* result) {
  if (strcmp(command, "FISH") == 0) {
    if (strcmp(result, "NONE") != 0) {
      addItemToStock(&player->stock);
    }
  }
  else if (strcmp(command, "SELL") == 0) {
      setZeroItemsOnStock(&player->stock);
```

```
  }
}


/**
 * @brief Create a Map object
 *
 * @return Map
 */
Map createMap() {
  Map map;

  scanf("AREA %i %i", &map.height, &map.width);

  return map;
}


/**
 * @brief Create a Player object
 *
 * @return Player
 */
Player createPlayer()
{
  Player player;

  scanf(" ID %s", player.id);

  setZeroItemsOnStock(&player.stock);
```

```
  return player;
}



/**
  * @brief Main function that brings together all other functions and
executions
 *
 * @return int
 */
int main() {
  char result[MAX_LINE];
  char* command = (char*) calloc(MAX_LINE, sizeof(char));

  setbuf(stdin, NULL);
  setbuf(stdout, NULL);
  setbuf(stderr, NULL);

  Map map = createMap();
  Player player = createPlayer();

  while (1) {
    read(&player, &map);
    command = chooseCommand(player, map);
    printf("%s\n", command);
    scanf("%s", result);
    react(&player, command, result);
```

```
    }


    free(map.points);
    free(command);


    return 0;
}
```