ESCOLA SUPERIOR DE TECNOLOGIA - IPS

ANO LETIVO 2015 / 2016

ENGENHARIA DE INFORMÁTICA

ALGORITMOS E TIPO ABSTRATOS DE DADOS

PROFª ROSSANA SANTOS

MINI-PROJETO 1

CAÇA AO TESOURO

MIGUEL FURTADO 120221006

Índice

Tipo de dados usados e respetiva Implementação

Stack Estático

Foram implementadas 3 stacks estáticos ao todo no desenvolvimento do projeto, sendo uma para os movimentos, uma para os tesouros e uma para as localizações.

Stack Estática Movimentos .H

```
#ifndef STACK_H_INCLUDED
#define STACK_H_INCLUDED

#define N 100
#include "elemento.h"

struct stack
{
   TElem table[N];
   unsigned int size;
};

typedef struct stack *PtStackE;

PtStackE stackCreateEstatico(void);
int stackDestroyEstatico(PtStackE *pstack);
int stackPushEstatico(PtStackE pstack, TElem pelem);
int stackPopEstatico(PtStackE pstack, TElem *pelem);
int stackPeekEstatico(PtStackE pstack, TElem *pelem);
int stackIsEmptyEstatico(PtStackE pstack);
int stackIsFullEstatico(PtStackE pstack);
int stackSizeEstatico(PtStackE pstack, unsigned int *pelement);

#endif // STACK_H_INCLUDED
```

Stack Estática Movimentos .C

```
#include <stdlib.h>
#include <stdio.h>
#include "stackEstatico.h"

#define OK         0
#define NULL_PTR    1
#define NO_STACK    2
#define STACK_EMPTY 4
#define STACK_FULL  5
```

```
PtStackE stackCreateEstatico()
{
   PtStackE myStack;

   if((myStack = (PtStackE) malloc (sizeof (struct stack))) == NULL)
      return NULL;

   myStack->size = 0;

   return myStack;
}

int stackDestroyEstatico(PtStackE *pstack)
{
   if(*pstack == NULL) return NO_STACK;
   free(*pstack);
   *pstack = NULL;

   return OK;
}

int stackPushEstatico(PtStackE pstack, TElem pelem)
{
   if(pstack == NULL) return NO_STACK;
   if(pstack->size == N) return STACK_FULL;

   pstack->table[pstack->size++] = pelem;
   return OK;
}

int stackPopEstatico(PtStackE pstack, TElem *pelem)
{
   if(pstack == NULL) return NO_STACK;
   if(pstack->size == 0) return STACK_EMPTY;
   if(pelem == NULL) return NULL_PTR;

   *pelem = pstack->table[--pstack->size];

   return OK;
}

int stackPeekEstatico(PtStackE pstack, TElem *pelem)
{
   if(pstack == NULL) return NO_STACK;
   if(pstack->size == 0) return STACK_EMPTY;
   if(pelem == NULL) return NULL_PTR;

   *pelem = pstack->table[pstack->size];
```

```
   return OK;
}

int stackIsEmptyEstatico(PtStackE pstack)
{
   if(pstack == NULL) return NO_STACK;
   if(pstack->size == 0) return STACK_EMPTY;

   return OK;
}

int stackIsFullEstatico(PtStackE pstack)
{
   if(pstack == NULL) return NO_STACK;
   if(pstack->size == N) return STACK_FULL;

    return OK;
}

int stackSizeEstatico(PtStackE pstack, unsigned int *pelement)
{
   if(pstack == NULL) return NO_STACK;
   if(pelement == NULL) return NULL_PTR;

   *pelement = pstack->size;
   if(pstack->size == 0) return STACK_EMPTY;

   return OK;
}
```

Stack Estática Tesouros .H

```
#ifndef STACKESTATICOTESOUROS_H_INCLUDED
#define STACKESTATICOTESOUROS_H_INCLUDED

#define N 100
#include "elemento2.h"

struct stackt
{
   TElemT table[N];
   unsigned int size;
};

typedef struct stackt *PtStackET;

PtStackET stackCreateEstaticoT(void);
int stackDestroyEstaticoT(PtStackET *pstack);
```

```
int stackPushEstaticoT(PtStackET pstack, TElemT pelem);
int stackPopEstaticoT(PtStackET pstack, TElemT *pelem);
int stackPeekEstaticoT(PtStackET pstack, TElemT *pelem);
int stackIsEmptyEstaticoT(PtStackET pstack);
int stackIsFullEstaticoT(PtStackET pstack);
int stackSizeEstaticoT(PtStackET pstack, unsigned int *pelement);

#endif // STACKESTATICOTESOUROS_H_INCLUDED
```

Stack Estática Tesouros .C

```
#include <stdlib.h>
#include <stdio.h>
#include "stackEstaticoTesouros.h"

#define OK         0
#define NULL_PTR    1
#define NO_STACK    2
#define STACK_EMPTY 4
#define STACK_FULL  5

PtStackET stackCreateEstaticoT()
{
   PtStackET myStack;

   if((myStack = (PtStackET) malloc (sizeof (struct stackt))) == NULL)
      return NULL;

   myStack->size = 0;

   return myStack;
}

int stackDestroyEstaticoT(PtStackET *pstack)
{
   if(*pstack == NULL) return NO_STACK;
   free(*pstack);
   *pstack = NULL;

   return OK;
}

int stackPushEstaticoT(PtStackET pstack, TElemT pelem)
{
   if(pstack == NULL) return NO_STACK;
   if(pstack->size == N) return STACK_FULL;

   pstack->table[pstack->size++] = pelem;
```

```
   return OK;
}

int stackPopEstaticoT(PtStackET pstack, TElemT *pelem)
{
   if(pstack == NULL) return NO_STACK;
   if(pstack->size == 0) return STACK_EMPTY;
   if(pelem == NULL) return NULL_PTR;

   *pelem = pstack->table[--pstack->size];

   return OK;
}

int stackPeekEstaticoT(PtStackET pstack, TElemT *pelem)
{
   if(pstack == NULL) return NO_STACK;
   if(pstack->size == 0) return STACK_EMPTY;
   if(pelem == NULL) return NULL_PTR;

   *pelem = pstack->table[pstack->size];
   return OK;
}

int stackIsEmptyEstaticoT(PtStackET pstack)
{
   if(pstack == NULL) return NO_STACK;
   if(pstack->size == 0) return STACK_EMPTY;

   return OK;
}

int stackIsFullEstaticoT(PtStackET pstack)
{
   if(pstack == NULL) return NO_STACK;
   if(pstack->size == N) return STACK_FULL;
   else return OK;
}

int stackSizeEstaticoT(PtStackET pstack, unsigned int *pelement)
{
   if(pstack == NULL) return NO_STACK;
   if(pelement == NULL) return NULL_PTR;

   *pelement = pstack->size;

   return OK;
}
```

Stack Estática Localizações .H

```
#ifndef STACKESTATICOLOCALIZACAO_H_INCLUDED
#define STACKESTATICOLOCALIZACAO_H_INCLUDED

#define N 100
#include "elemento3.h"

struct stackl
{
   TElemL table[N];
   unsigned int size;
};

typedef struct stackl *PtStackEL;

PtStackEL stackCreateEstaticoL(void);
int stackDestroyEstaticoL(PtStackEL *pstack);
int stackPushEstaticoL(PtStackEL pstack, TElemL pelem);
int stackPopEstaticoL(PtStackEL pstack, TElemL *pelem);
int stackPeekEstaticoL(PtStackEL pstack, TElemL *pelem);
int stackIsEmptyEstaticoL(PtStackEL pstack);
int stackIsFullEstaticoL(PtStackEL pstack);
int stackSizeEstaticoL(PtStackEL pstack, unsigned int *pelement);

#endif // STACKESTATICOLOCALIZACAO_H_INCLUDED
```

Stack Estática Localizações .C

```
#include <stdlib.h>
#include <stdio.h>
#include "stackEstaticoLocalizacao.h"

#define OK         0
#define NULL_PTR    1
#define NO_STACK    2
#define STACK_EMPTY 4
#define STACK_FULL  5


PtStackEL stackCreateEstaticoL()
{
   PtStackEL myStack;

   if((myStack = (PtStackEL) malloc (sizeof (struct stackl))) == NULL)
      return NULL;

   myStack->size = 0;
```

```
   return myStack;
}

int stackDestroyEstaticoL(PtStackEL *pstack)
{
   if(*pstack == NULL) return NO_STACK;
   free(*pstack);
   *pstack = NULL;

   return OK;
}

int stackPushEstaticoL(PtStackEL pstack, TElemL pelem)
{
   if(pstack == NULL) return NO_STACK;
   if(pstack->size == N) return STACK_FULL;

   pstack->table[pstack->size++] = pelem;
   return OK;
}

int stackPopEstaticoL(PtStackEL pstack, TElemL *pelem)
{
   if(pstack == NULL) return NO_STACK;
   if(pstack->size == 0) return STACK_EMPTY;
   if(pelem == NULL) return NULL_PTR;

   *pelem = pstack->table[--pstack->size];

   return OK;
}

int stackPeekEstaticoL(PtStackEL pstack, TElemL *pelem)
{
   if(pstack == NULL) return NO_STACK;
   if(pstack->size == 0) return STACK_EMPTY;
   if(pelem == NULL) return NULL_PTR;

   *pelem = pstack->table[pstack->size];
   return OK;
}

int stackIsEmptyEstaticoL(PtStackEL pstack)
{
   if(pstack == NULL) return NO_STACK;
   if(pstack->size == 0) return STACK_EMPTY;
   else return OK;
```

```
}

int stackIsFullEstaticoL(PtStackEL pstack)
{
   if(pstack == NULL) return NO_STACK;
   if(pstack->size == N) return STACK_FULL;

   return OK;
}

int stackSizeEstaticoL(PtStackEL pstack, unsigned int *pelement)
{
   if(pstack == NULL) return NO_STACK;
   if(pelement == NULL) return NULL_PTR;

   *pelement = pstack->size;

   return OK;
}
```

Stack Dinâmico

Foram implementadas 3 stacks dinâmicas ao todo no desenvolvimento do projeto, sendo uma para os movimentos, uma para os tesouros e uma para as localizações.

Stack Dinâmico Movimentos .H

```
#ifndef STACKDINAMICO_H_INCLUDED
#define STACKDINAMICO_H_INCLUDED

#include "elemento.h"

typedef struct node *PtNode;

struct node
{
   TElem *ptElem;
   PtNode ptPrev;
};

 struct stackd
{
   PtNode top;
   unsigned int size;
};

typedef struct stackd *PtStackD;

PtStackD stackCreateDinamico(void);
```

```
int stackDestroyDinamico(PtStackD *pStack);
int stackPushDinamico(PtStackD pStack, TElem pelem);
int stackPopDinamico(PtStackD pStack, TElem *pelem);
int stackPeekDinamico(PtStackD pStack, TElem *pelem);
int stackIsEmptyDinamico(PtStackD pStack);
int stackIsFullDinamico(PtStackD pStack);
int stackSizeDinamico(PtStackD pStack, unsigned int *pnelem);

#endif // STACKDINAMICO_H_INCLUDED
```

Stack Dinâmico Movimentos .C

```
#include <stdio.h>
#include <stdlib.h>
#include "stackDinamico.h"

#define OK         0
#define NULL_PTR    1
#define NULL_SIZE   2
#define NO_MEM      3
#define STACK_EMPTY 4
#define NO_STACK    5

PtStackD stackCreateDinamico()
{
   PtStackD myStack;

   if((myStack = (PtStackD) malloc (sizeof (struct stackd))) == NULL)
   {
      return NULL;
   }

   myStack->top = NULL;
   myStack->size = 0;

   return myStack;
}

int stackPushDinamico(PtStackD pStack, TElem pelem)
{
   PtNode tmpNode;

   if(pStack == NULL) return NO_STACK;
   if((tmpNode = (PtNode)malloc(sizeof (struct node))) == NULL)return NO_MEM;
   if((tmpNode->ptElem = (TElem *)malloc (sizeof (TElem))) == NULL)
   {
      free(tmpNode);
      return NO_MEM;
```

```
    }

    *tmpNode->ptElem = pelem;
    tmpNode->ptPrev = pStack->top;

    pStack->top = tmpNode;
    pStack->size++;

    return OK;
}

int stackPopDinamico(PtStackD pStack, TElem *pelem)
{
    PtNode tmpNode;

    if(pStack == NULL)return NO_STACK;
    if(pStack->top == NULL)return STACK_EMPTY;
    if(pelem == NULL)return NULL_PTR;


    *pelem = *pStack->top->ptElem;
    tmpNode = pStack->top;
    pStack->top = pStack->top->ptPrev;
    pStack->size--;
    free(tmpNode->ptElem);
    free(tmpNode);

    return OK;
}

int stackDestroyDinamico(PtStackD *ptStack)
{
    PtStackD tmpStack = *ptStack;
    PtNode tmpNode;

    if(tmpStack == NULL)return NO_STACK;

    while(tmpStack->top != NULL)
    {
        tmpNode = tmpStack->top;
        tmpStack->top = tmpStack->top->ptPrev;
        free(tmpNode->ptElem);
        free(tmpNode);
    }

    free(tmpStack);
    *ptStack = NULL;
```

```
   return OK;
}

int stackPeekDinamico(PtStackD pStack, TElem *pelem)
{
   if(pStack == NULL)
   {
      return NO_STACK;

   }
   if(pStack->top == NULL)
   {
      return STACK_EMPTY;
   }
   if(pelem == NULL)
   {
      return NULL_PTR;
   }

   *pelem = *pStack->top->ptElem;

   return OK;
}

int stackIsEmptyDinamico (PtStackD pStack)
{

   if(pStack->top == NULL)
   {
      return 1;
   }
   return 0;
}

int stackSizeDinamico(PtStackD pstack, unsigned int *pnelem)
{
   if(pstack == NULL)
   {
      return NO_STACK;
   }
   if(pnelem == NULL)
   {
      return NULL_PTR;
   }

   *pnelem = pstack->size;
   return OK;
}
```

Stack Dinâmico Tesouros .H

```
#ifndef STACKDINAMICOTESOUROS_H_INCLUDED
#define STACKDINAMICOTESOUROS_H_INCLUDED
#include "elemento2.h"

typedef struct nodedt *PtNodeDT;

struct nodedt
{
   TElemT *ptElem;
   PtNodeDT ptPrev;
};

 struct stackdt
{
   PtNodeDT top;
   unsigned int size;
};

typedef struct stackdt *PtStackDT;

PtStackDT stackCreateDinamicoT(void);
int stackDestroyDinamicoT(PtStackDT *pStack);
int stackPushDinamicoT(PtStackDT pStack, TElemT pelem);
int stackPopDinamicoT(PtStackDT pStack, TElemT *pelem);
int stackPeekDinamicoT(PtStackDT pStack, TElemT *pelem);
int stackIsEmptyDinamicoT(PtStackDT pStack);
int stackIsFullDinamicoT(PtStackDT pStack);
int stackSizeDinamicoT(PtStackDT pStack, unsigned int *pnelem);

#endif // STACKDINAMICOTESOUROS_H_INCLUDED
```

Stack Dinâmico Tesouros .C

```
#include <stdio.h>
#include <stdlib.h>
#include "stackDinamicoTesouros.h"

#define OK         0
#define NULL_PTR    1
#define NULL_SIZE   2
#define NO_MEM      3
#define STACK_EMPTY 4
#define NO_STACK    5

PtStackDT stackCreateDinamicoT()
{
```

```
   PtStackDT myStack;

   if((myStack = (PtStackDT) malloc (sizeof (struct stackdt))) == NULL)
   {
      return NULL;
   }

   myStack->top = NULL;
   myStack->size = 0;

   return myStack;
}

int stackPushDinamicoT(PtStackDT pStack, TElemT pelem)
{
   PtNodeDT tmpNode;

   if(pStack == NULL) return NO_STACK;
   if((tmpNode = (PtNodeDT)malloc(sizeof (struct nodedt))) == NULL)return NO_MEM;
   if((tmpNode->ptElem = (TElemT *)malloc (sizeof (TElemT))) == NULL)
   {
      free(tmpNode);
      return NO_MEM;
   }

   *tmpNode->ptElem = pelem;
   tmpNode->ptPrev = pStack->top;

   pStack->top = tmpNode;
   pStack->size++;

   return OK;
}

int stackPopDinamicoT(PtStackDT pStack, TElemT *pelem)
{
   PtNodeDT tmpNode;

   if(pStack == NULL)return NO_STACK;

   if(pStack->top == NULL)return STACK_EMPTY;

   if(pelem == NULL)return NULL_PTR;


   *pelem = *pStack->top->ptElem;
   tmpNode = pStack->top;
   pStack->top = pStack->top->ptPrev;
```

```
    pStack->size--;
    free(tmpNode->ptElem);
    free(tmpNode);

    return OK;
}

int stackDestroyDinamicoT(PtStackDT *ptStack)
{
    PtStackDT tmpStack = *ptStack;
    PtNodeDT tmpNode;

    if(tmpStack == NULL)return NO_STACK;

    while(tmpStack->top != NULL)
    {
        tmpNode = tmpStack->top;
        tmpStack->top = tmpStack->top->ptPrev;
        free(tmpNode->ptElem);
        free(tmpNode);
    }

    free(tmpStack);
    *ptStack = NULL;

    return OK;
}

int stackPeekDinamicoT(PtStackDT pStack, TElemT *pelem)
{
    if(pStack == NULL)
    {
        return NO_STACK;

    }
    if(pStack->top == NULL)
    {
        return STACK_EMPTY;
    }
    if(pelem == NULL)
    {
        return NULL_PTR;
    }

    *pelem = *pStack->top->ptElem;

    return OK;
}
```

```
int stackIsEmptyDinamicoT (PtStackDT pStack)
{

   if(pStack->top == NULL)
   {
      return 1;
   }
   return 0;
}

int stackSizeDinamicoT(PtStackDT pstack, unsigned int *pnelem)
{
   if(pstack == NULL)
   {
      return NO_STACK;
   }
   if(pnelem == NULL)
   {
      return NULL_PTR;
   }

   *pnelem = pstack->size;

   return OK;
}
```

Stack Dinâmico Localizações .H

```
#ifndef STACKDINAMICOLOCALIZACAO_H_INCLUDED
#define STACKDINAMICOLOCALIZACAO_H_INCLUDED
#include "elemento3.h"

typedef struct nodedl *PtNodeDL;

struct nodedl
{
   TElemL *ptElem;
   PtNodeDL ptPrev;
};

 struct stackdl
{
   PtNodeDL top;
   unsigned int size;
};

typedef struct stackdl *PtStackDL;
```

```
PtStackDL stackCreateDinamicoL(void);
int stackDestroyDinamicoL(PtStackDL *pStack);
int stackPushDinamicoL(PtStackDL pStack, TElemL pelem);
int stackPopDinamicoL(PtStackDL pStack, TElemL *pelem);
int stackPeekDinamicoL(PtStackDL pStack, TElemL *pelem);
int stackIsEmptyDinamicoL(PtStackDL pStack);
int stackIsFullDinamicoL(PtStackDL pStack);
int stackSizeDinamicoL(PtStackDL pStack, unsigned int *pnelem);

#endif // STACKDINAMICOLOCALIZACAO_H_INCLUDED
```

Stack Dinâmico Localizações .C

```
#include <stdio.h>
#include <stdlib.h>
#include "stackDinamicoLocalizacao.h"

#define OK         0
#define NULL_PTR    1
#define NULL_SIZE   2
#define NO_MEM      3
#define STACK_EMPTY 4
#define NO_STACK    5

PtStackDL stackCreateDinamicoL()
{
   PtStackDL myStack;

   if((myStack = (PtStackDL) malloc (sizeof (struct stackdl))) == NULL)
   {
      return NULL;
   }

   myStack->top = NULL;
   myStack->size = 0;

   return myStack;
}

int stackPushDinamicoL(PtStackDL pStack, TElemL pelem)
{
   PtNodeDL tmpNode;

   if(pStack == NULL) return NO_STACK;
   if((tmpNode = (PtNodeDL)malloc(sizeof (struct nodedl))) == NULL)return NO_MEM;
   if((tmpNode->ptElem = (TElemL *)malloc (sizeof (TElemL))) == NULL)
   {
```

```
        free(tmpNode);
        return NO_MEM;
    }

    *tmpNode->ptElem = pelem;
    tmpNode->ptPrev = pStack->top;

    pStack->top = tmpNode;
    pStack->size++;

    return OK;
}

int stackPopDinamicoL(PtStackDL pStack, TElemL *pelem)
{
    PtNodeDL tmpNode;

    if(pStack == NULL)return NO_STACK;

    if(pStack->top == NULL)return STACK_EMPTY;

    if(pelem == NULL)return NULL_PTR;


    *pelem = *pStack->top->ptElem;
    tmpNode = pStack->top;
    pStack->top = pStack->top->ptPrev;
    pStack->size--;
    free(tmpNode->ptElem);
    free(tmpNode);

    return OK;
}

int stackDestroyDinamicoL(PtStackDL *ptStack)
{
    PtStackDL tmpStack = *ptStack;
    PtNodeDL tmpNode;

    if(tmpStack == NULL)return NO_STACK;

    while(tmpStack->top != NULL)
    {
        tmpNode = tmpStack->top;
        tmpStack->top = tmpStack->top->ptPrev;
        free(tmpNode->ptElem);
        free(tmpNode);
    }
```

```
    free(tmpStack);
    *ptStack = NULL;

    return OK;
}

int stackPeekDinamicoL(PtStackDL pStack, TElemL *pelem)
{
    if(pStack == NULL)
    {
        return NO_STACK;

    }
    if(pStack->top == NULL)
    {
        return STACK_EMPTY;
    }
    if(pelem == NULL)
    {
        return NULL_PTR;
    }

    *pelem = *pStack->top->ptElem;

    return OK;
}

int stackIsEmptyDinamicoL (PtStackDL pStack)
{

    if(pStack->top == NULL)
    {
        return 1;
    }
    return 0;
}

int stackSizeDinamicoL(PtStackDL pstack, unsigned int *pnelem)
{
    if(pstack == NULL)
    {
        return NO_STACK;
    }
    if(pnelem == NULL)
    {
        return NULL_PTR;
    }
```

```
   *pnelem = pstack->size;

   return OK;
}
```

Queue Estático

Para o desenvolvimento do projeto foi apenas desenvolvida uma queue estática com o intuito de guardar os movimentos para que posteriormente fossem usados no desenrolar do jogo.

Queue Estático Movimentos .H

```
#ifndef FILAESTATICA_H_INCLUDED
#define FILAESTATICA_H_INCLUDED
#define N 100
#include "elemento.h"

struct queue
{
   TElem table[N];
   unsigned int begin, end;
   unsigned int nElem;
};

typedef struct queue *PtQueue;

PtQueue queueCreateEstatico(void);
int queueDestroyEstatico (PtQueue *pQueue);
int queueEnqueueEstatico(PtQueue pQueue, TElem pelem);
int queueDequeueEstatico(PtQueue pQueue, TElem *pelem);
int queuePeekEstatico(PtQueue pQueue, TElem *pelem);
int queueIsEmptyEstatico(PtQueue pQueue);
int queueIsFullEstatico(PtQueue pQueue);
int queueSizeEstatico(PtQueue pQueue, unsigned int *pNElem);

#endif // QUEUE_H_INCLUDED
```

Queue Estático Movimentos .C

```
#include <stdlib.h>
#include <stdio.h>
#include "filaEstatica.h"

#define OK         0
#define NO_QUEUE    1
#define NO_MEM      2
#define NULL_PTR    3
```

```
#define QUEUE_EMPTY 4
#define QUEUE_FULL  5

PtQueue queueCreateEstatico(void)
{
   PtQueue queue;

   if((queue = (PtQueue) malloc (sizeof (struct queue))) == NULL)
   {
      return NULL;
   }
   queue->begin = 0;
   queue->end = 0;
   queue->nElem = 0;

   return queue;
}

int queueDestroyEstatico (PtQueue *pQueue)
{
   PtQueue TmpQueue = *pQueue;

   if(TmpQueue == NULL) return NO_QUEUE;

   free(TmpQueue);
   *pQueue = NULL;

   return OK;
}
int queueEnqueueEstatico(PtQueue pQueue, TElem pelem)
{
   if(pQueue == NULL) return NO_QUEUE;
   if(queueIsFullEstatico(pQueue)) return QUEUE_FULL;

   pQueue->table[pQueue->end] = pelem;
   pQueue->end = (pQueue->end + 1) % N;
   pQueue->nElem++;

   return OK;
}

int queueDequeueEstatico(PtQueue pQueue, TElem *pelem)
{
   if(pQueue == NULL) return NO_QUEUE;
   if(queueIsEmptyEstatico(pQueue)) return QUEUE_EMPTY;
   if(pelem == NULL) return NULL_PTR;

   *pelem = pQueue->table[pQueue->begin];
```

```
   pQueue->begin = (pQueue->begin + 1) % N;
   pQueue->nElem--;

   return OK;
}
int queuePeekEstatico(PtQueue pQueue, TElem *pelem)
{
   if(pQueue == NULL) return NO_QUEUE;
   if(queueIsEmptyEstatico(pQueue)) return QUEUE_EMPTY;
   if(pelem == NULL) return NULL_PTR;

   *pelem = pQueue->table[pQueue->begin];

   return OK;
}
int queueIsEmptyEstatico(PtQueue pQueue)
{
   if(pQueue == NULL) return NO_QUEUE;
   if(pQueue->nElem == 0)
   {
      return 1;
   }
   else
   {
      return 0;
   }
}
int queueIsFullEstatico(PtQueue pQueue)
{
   if(pQueue == NULL) return NO_QUEUE;
   if(pQueue->nElem == N)
   {
      return 1;
   }
   else
   {
      return 0;
   }
}
int queueSizeEstatico(PtQueue pQueue, unsigned int *pNElem)
{
   if(pQueue == NULL) return NO_QUEUE;
   if(pNElem == NULL) return NULL_PTR;

   *pNElem = pQueue->nElem;

   return OK;
}
```

Queue Dinâmico

Para o desenvolvimento do projeto foi apenas desenvolvida uma queue dinâmica com o intuito de guardar os movimentos para que posteriormente fossem usados no desenrolar do jogo.

Queue Dinâmico Movimentos .H

```
#ifndef FILADINAMICA_H_INCLUDED
#define FILADINAMICA_H_INCLUDED
#define N 100
#include "elemento.h"

typedef struct noded *PtNodeD;

struct noded
{
   TElem *ptElem;
   PtNodeD ptNext;
};

struct queued
{
   PtNodeD head, tail;
   unsigned int nElems;
};

typedef struct queued *PtQueueD;

PtQueueD queueCreateDinamico(void);
int queueDestroyDinamico (PtQueueD *pQueue);
int queueEnqueueDinamico(PtQueueD pQueue, TElem pelem);
int queueDequeueDinamico(PtQueueD pQueue, TElem *pelem);
int queuePeekDinamico(PtQueueD pQueue, TElem *pelem);
int queueIsEmptyDinamico(PtQueueD pQueue);
int queueSizeDinamico(PtQueueD pQueue, unsigned int *pNElem);

#endif // FILADINAMICA_H_INCLUDED
```

Queue Dinâmico Movimentos .C

```
#include <stdio.h>
#include <stdlib.h>
#include "filaDinamica.h"

#define OK        0
```

```
#define NO_QUEUE    1
#define NO_MEM      2
#define NULL_PTR    3
#define QUEUE_EMPTY 4
#define QUEUE_FULL  5

PtQueueD queueCreateDinamico(void)
{
   PtQueueD queue;

   if((queue = (PtQueueD) malloc(sizeof(struct queued))) == NULL)
      return NULL;

   queue->head = NULL;
   queue->tail = NULL;
   queue->nElems = 0;

   return queue;
}

int queueDestroyDinamico (PtQueueD *pQueue)
{
   PtQueueD tmpQueue = *pQueue;
   PtNodeD tmpNode;
   if(tmpQueue == NULL) return NO_QUEUE;

   while(tmpQueue->head != NULL)
   {
      tmpNode = tmpQueue->head;
      tmpQueue->head = tmpQueue->head->ptNext;
      free(tmpNode->ptElem);
      free(tmpNode);
   }

   free(tmpQueue);
   *pQueue = NULL;

   return OK;
}

int queueEnqueueDinamico(PtQueueD pQueue, TElem pelem)
{
   PtNodeD tmpNode;

   if(pQueue == NULL) return NO_QUEUE;
   if((tmpNode = (PtNodeD) malloc (sizeof (struct noded))) == NULL) return NO_MEM;
   if((tmpNode->ptElem = (TElem *) malloc (sizeof (TElem))) == NULL)
   {
```

```
        free(tmpNode);
        return NO_MEM;
    }

    tmpNode->ptNext = NULL;

    if(pQueue->tail == NULL)pQueue->head = tmpNode;
    else pQueue->tail->ptNext = tmpNode;

    pQueue->tail = tmpNode;
    *pQueue->tail->ptElem = pelem;
    pQueue->nElems++;

    return OK;
}

int queueDequeueDinamico(PtQueueD pQueue, TElem *pelem)
{
    PtNodeD tmpNode;

    if(pQueue == NULL) return NO_QUEUE;
    if(pQueue->head == NULL) return QUEUE_EMPTY;
    if(pelem == NULL) return NULL_PTR;

    *pelem = *pQueue->head->ptElem;
    tmpNode = pQueue->head;
    pQueue->head = pQueue->head->ptNext;

    if(pQueue->head == NULL) pQueue->tail = NULL;

    free(tmpNode->ptElem);
    free(tmpNode);

    pQueue->nElems--;

    return OK;
}
int queuePeekDinamico(PtQueueD pQueue, TElem *pelem)
{
    if(pQueue == NULL) return NO_QUEUE;
    if(pQueue->head == NULL) return QUEUE_EMPTY;
    if(pelem == NULL) return NULL_PTR;

    *pelem = *pQueue->head->ptElem;

    return OK;
}
int queueIsEmptyDinamico(PtQueueD pQueue)
```

```
{
   return (pQueue == NULL || pQueue->head == NULL);
}

int queueSizeDinamico(PtQueueD pQueue, unsigned int *pNElem)
{
   if(pQueue == NULL) return NO_QUEUE;
   if(pQueue->head == NULL) return QUEUE_EMPTY;
   if(pNElem == NULL) return NULL_PTR;

   int counter = 0;
   PtNodeD tmpNode = pQueue->head;

   while(tmpNode != NULL)
   {
      counter++;
      tmpNode = tmpNode->ptNext;
   }

   *pNElem = counter;
   return OK;
}
```

Complexidade Algorítmica

Procura de Tesouros

Para a procura de tesouros cada vez que é feito um movimento, seja este um Move, Loop ou Jump, cada vez que se passa por uma localização vai ser confirmado se naquela localização em especifico existe um tesouro, como tal foi criado o método existeTesouro que vai fazer essa confirmação.

O método existeTesouro o que faz é ir ver se dentro da pilha de tesouros a localização onde se encontra atualmente é um tesouro com a ajuda de uma pilha auxiliar. Entra num ciclo while e primeiro faz pop da pilha de tesouros e vai verificar se é igual à localização atual. Se encontrar um tesouro nessa localização muda a variável encontrado no tesouro para 1, incrementa o número de tesouros encontrados para mais 1 e guarda o tesouro numa pilha auxiliar. Caso a localização atual tenha 1 tesouro mas já tenha sido encontrado apenas faz push para a pilha auxiliar. Se a localização do tesouro não for igual à localização atual apenas faz push para a pilha auxiliar e volta a fazer as verificações para o próxima tesouro na pilha de tesouros até esta estar vazia. Quando a pilha de tesouros está vazia passa para outro ciclo while em que simplesmente faz pop da pilha auxiliar e push para a pilha de tesouros até a pilha auxiliar ficar vazia, para que na próxima vez que o método for chamado consiga usar a pilha de tesouros com todos os tesouros. Assim ficam calculados quantos tesouros foram encontrados no fim do jogo.

Este algoritmo é de complexidade **O(n)** pois vai depender do número de tesouros que existe na pilha que pode variar consoante o ficheiro de configuração que é lido no inicio.

Cálculo da Distância Percorrida

Para o cálculo da distância percorrida o que fiz foi simplesmente ir buscar o número de elementos que se encontravam na pilha de localizações. Com o desenrolar do jogo todas as localizações por onde passou foram guardadas numa pilha, como cada localização vale 1 ponto então concluí que o cálculo da distância percorrida é igual ao número de elementos que se encontra na pilha de localizações.

A complexidade algorítmica é de **O(I)** pois a única coisa que é usada é o size do stack.

Contagem das Localizações

A contagem das localizações foi onde surgiram mais dificuldades, e como tal foram precisos criar 3 métodos, o método contarLocalizacoes que vai ser a junção de os 3 métodos para obter as localizações com as respetivas contagens, o método contador onde conta o número de aparições de cada localização e o método confirm onde vai limpar a stack de localizações com o intuito de apenas conter as localizações onde se passou mais que 1 vez.

No primeiro método, contarLocalizacoes, foi criado uma pilha auxiliar de localizações, uma estrutura localizações e um unsigned int nElem. Ométodo começa por guardar o tamanho do stack de localizações que é recebido como argumento no método contarLocalizacoes. O tamanho vai servir para que ao entrar no while cada elemento vá ser contado individualmente em quantas vezes aparece dentro da stack de localizações. Entra no primeiro while e como existem elementos na stack passa para o segundo while onde tira um elemento da stack e vai entra no método contador. Depois de sair do contador é feito um push para stack auxiliar já com a contagem de quantas aparições essa localização tem no stack de localizações. Após ter feito este ciclo em todos os elementos passa para um terceiro while que só acaba quando tira todos os elementos da stack auxiliar que têm o número de aparições maior que 2 e põe na stack de localizações. Por fim é destruída a pilha auxiliar e entra no método confirm para se tirar as localizações repetidas.

No segundo método, contador, é criada uma stack auxiliar e uma estrutura de localizacaoAtual. Nesta estrutura define-se o n a 0, pois inicialmente não se sabe quantas se passou na mesma localização, define-se o x e o y da estrutura para ser igual à localização enviada como argumento e faz-se um push da localização recebida como argumento para a pilha auxiliar. Entra-se num ciclo while e este vai verificar se dentro da pilha de localizações quantas vezes é que cada localização aparece e sempre que aparecer incrementa-se mais 1 na variável n associada à estrutura de localizacaoAtual. Após este ciclo ter terminado entra-se noutro ciclo em que se atribui a essas localizações que apareceram o n. Destrói-se a stack auxiliar e volta ao método contarLocalizações.

No terceiro método, confirm, foram criados dois stacks auxiliares, uma estrutura localizacaoAtual, uma estrutura localização e um unsigned int nElem. Inicialmente entra-se num ciclo while que vai ler enquanto o número de elementos disponíveis na pilha for mais que 0. Ao se entrar nesta while faz-se pop do primeiro elemento da stack de localizações e põe-se na pilha aux2 e faz-se menos 1 no nElem, assim o ciclo sabe que já tem menos um elemento que vai ter que confirmar e ao guardar na aux2 mais tarde pode-se ir buscar este elemento. Define-se o x e o y da estrutura localizacaoAtual para ser igual à localização retirada da stack e de seguida entra-se no segundo ciclo while. Este ciclo vai funcionar enquanto a stack tiver elementos, pois o objetivo é que o ciclo vá verificar se existem localizações igual à primeira e se houver retirar da pilha pois já não são precisas. Tendo isto em conta, sempre que encontra um elemento

decrementa 1 do número de elementos pois na stack de localizações agora tem menos 1 elemento para verificar, caso não encontre um elemento igual ao inicial simplesmente guarda no stack aux. Depois de verificar todos os elementos da stack passa para um segundo while onde vai por todos os elementos que estão na stack aux de novo na stack de localizações que neste momento vai ter apenas os elementos que ainda não foram confirmados se havia repetidos. Depois de estes 2 ciclos serem concluídos o ciclo onde eles se inserem também é concluído. Entra-se num novo ciclo que o objetivo é pegar na stack aux2 que contém apenas as localizações iniciais com as contagens já feitas e passa para a stack de localizações. Por fim destrói-se as duas pilhas auxiliares pois já não são precisas e volta-se ao método contarLocalizações.

Este é o algoritmo mais complexo que desenvolvi para este projeto e dado que são vários métodos e ciclos encadeados, como descrito anteriormente, a complexidade algorítmica é de **O(n2)**, ou seja, quadrática.

Conclusão

Com o desenvolvimento deste mini-projeto foi-me possível aplicar todos os conteúdos lecionados em aula e apesar das dificuldades sentidas foram aplicados com sucesso.

Algumas das dificuldades sentidas passaram inicialmente por ler dos ficheiros e conseguir transformar as strings que eram lidas para um inteiro, um exemplo foi o caso do tamanho do mapa em que foi preciso usar 2 buffers para passar com o uso da função atoi para um inteiro de modo a ficar correto. Ler o ficheiro de movimentos foi também um desafio pois foi preciso decompô-los tendo em conta o undo para que no final ficassem os movimentos corretos a ser executados.

Uma grande dificuldade que penso ter sido das mais complicadas de resolver foi na soma das localizações. Independentemente ter sido um sucesso esta fez com que o algoritmo desenvolvido não fosse o mais eficiente a nível de performance.

Uma outra grande dificuldade foi na implementação das stacks e das queues, pois havia um erro na implementação, mas que com o desenrolar do desenvolvimento do projeto foi resolvido e ficou totalmente funcional.

Em suma concluo que o projeto foi desenvolvido com sucesso, como referido anteriormente e com isto no futuro irá ser possível desenvolver qualquer tipo de projetos que envolvam stack e queues sem problemas.