



A-level
COMPUTER SCIENCE
(7517/1C)

Skeleton Program

```
Program Monster;

{$IFDEF FPC}
  {$MODE Delphi}
{$ENDIF}
{$APPTYPE CONSOLE}
{$R *.res}

Uses
  SysUtils,
  UMonsterClasses in 'UMonsterClasses.pas';

Var
  Choice : Integer;
  MyGame : Game;

Procedure DisplayMenu;
Begin
  Writeln('MAIN MENU');
  Writeln;
  Writeln('1. Start new game');
  Writeln('2. Play training game');
  Writeln('9. Quit');
  Writeln;
  Write('Please enter your choice: ');
End;

Function GetMainMenuChoice: Integer;
Var
  Choice: Integer;
Begin
  Readln(Choice);
  Writeln;
  GetMainMenuChoice := Choice;
End;

Begin
  While Choice <> 9 Do
  Begin
    DisplayMenu;
    Choice := GetMainMenuChoice;
    Case Choice Of
      1:
        Begin
          MyGame := Game.Create(False);
          MyGame.Free;
        End;
      2:
        Begin
          MyGame := Game.Create(True);
          MyGame.Free;
        End;
    End;
  End;
End;
```

```
End;  
End.
```

```
Unit UMonsterClasses;  
  
{ $IFDEF FPC  
  { $MODE Delphi  
  { $ENDIF  
  
Interface  
  
Type  
  CellReference = Record  
    NoOfCellsEast, NoOfCellsSouth : Integer;  
  End;  
  
  Item = Class  
  Strict Protected  
    NoOfCellsEast, NoOfCellsSouth : Integer;  
  Public  
    Function GetPosition : CellReference;  
    Procedure SetPosition(Position : CellReference);  
    Function CheckIfSameCell(Position : CellReference) : Boolean;  
  End;  
  
  Character = Class(Item)  
  Public  
    Procedure MakeMove(Direction : Char);  
  End;  
  
  Enemy = Class(Item)  
  Strict Private  
    Awake : Boolean;  
  Public  
    Constructor Create;  
    Procedure MakeMove(PlayerPosition : CellReference); Virtual;  
    Function GetAwake : Boolean;  
    Procedure ChangeSleepStatus; Virtual;  
  End;  
  
  Trap = Class(Item)  
  Strict Private  
    Triggered : Boolean;  
  Public  
    Constructor Create;  
    Function GetTriggered : Boolean;  
    Procedure ToggleTrap;  
  End;  
  
  Grid = Class  
  Strict Private  
    NS, WE : Integer;
```

```
    CavernState : Array Of Array Of Char;
Public
    Constructor Create(S, E : Integer);
    Procedure Reset;
    Procedure Display(MonsterAwake : Boolean);
    Procedure PlaceItem(Position : CellReference; Item : Char);
    Function IsCellEmpty(Position : CellReference) : Boolean;
End;

Game = Class
Const
    NS = 4;
    WE = 6;
Strict Private
    Player : Character;
    Cavern : Grid;
    Monster : Enemy;
    Flask : Item;
    Trap1, Trap2 : Trap;
    TrainingGame : Boolean;
Public
    Constructor Create(IsATrainingGame : Boolean);
    Destructor Destroy; Override;
    Procedure Play;
    Procedure DisplayMoveOptions;
    Function GetMove : char;
    Procedure DisplayWonGameMessage;
    Procedure DisplayTrapMessage;
    Procedure DisplayLostGameMessage;
    Function CheckValidMove(Direction : Char) : Boolean;
    Function SetPositionOfItem(Item : Char) : CellReference;
    Procedure SetUpGame;
    Function GetNewRandomPosition: CellReference;
End;

Implementation

Function Item.CheckIfSameCell(Position : CellReference) : Boolean;
Begin
    If (NoOfCellsEast = Position.NoOfCellsEast) And
        (NoOfCellsSouth = Position.NoOfCellsSouth) Then
        CheckIfSameCell := True
    Else
        CheckIfSameCell := False;
End;

Function Item.GetPosition : CellReference;
Var
    Position: CellReference;
Begin
    Position.NoOfCellsEast := NoOfCellsEast;
    Position.NoOfCellsSouth := NoOfCellsSouth;
    GetPosition := Position;
End;
```

```
End;

Procedure Item.SetPosition(Position : CellReference);
Begin
  NoOfCellsEast := Position.NoOfCellsEast;
  NoOfCellsSouth := Position.NoOfCellsSouth;
End;

Procedure Character.MakeMove(Direction : Char);
Begin
  Case Direction Of
    'N':
      NoOfCellsSouth := NoOfCellsSouth - 1;
    'S':
      NoOfCellsSouth := NoOfCellsSouth + 1;
    'W':
      NoOfCellsEast := NoOfCellsEast - 1;
    'E':
      NoOfCellsEast := NoOfCellsEast + 1;
  End;
End;

Procedure Enemy.ChangeSleepStatus;
Begin
  Awake := Not Awake;
End;

Constructor Enemy.Create;
Begin
  Inherited;
  Awake := False;
End;

Function Enemy.GetAwake : Boolean;
Begin
  GetAwake := Awake;
End;

Procedure Enemy.MakeMove(PlayerPosition : CellReference);
Begin
  If NoOfCellsSouth < PlayerPosition.NoOfCellsSouth Then
    NoOfCellsSouth := NoOfCellsSouth + 1
  Else If NoOfCellsSouth > PlayerPosition.NoOfCellsSouth Then
    NoOfCellsSouth := NoOfCellsSouth - 1
  Else If NoOfCellsEast < PlayerPosition.NoOfCellsEast Then
    NoOfCellsEast := NoOfCellsEast + 1
  Else
    NoOfCellsEast := NoOfCellsEast - 1;
End;

Constructor Trap.Create;
Begin
  Inherited;
  Triggered := False;
```

```
End;

Function Trap.GetTriggered : Boolean;
Begin
  GetTriggered := Triggered;
End;

Procedure Trap.ToggleTrap;
Begin
  Triggered := Not Triggered;
End;

Procedure Grid.Display(MonsterAwake : Boolean);
Var
  Count1, Count2 : Integer;
Begin
  For Count1 := 0 To NS Do
    Begin
      Writeln(' -----');
      For Count2 := 0 To WE Do
        If (CavernState[Count1, Count2] = ' ') Or
            (CavernState[Count1, Count2] = '*') Or
            ((CavernState[Count1, Count2] = 'M') And MonsterAwake) Then
          Write('|' + CavernState[Count1, Count2])
        Else
          Write('| ');
      Writeln('|');
    End;
    Writeln(' ----- ');
  Writeln;
End;

Function Grid.IsCellEmpty(Position : CellReference) : Boolean;
Begin
  If CavernState[Position.NoOfCellsSouth, Position.NoOfCellsEast] = ' ' Then
    IsCellEmpty := True
  Else
    IsCellEmpty := False;
End;

Procedure Grid.PlaceItem(Position : CellReference; Item : Char);
Begin
  CavernState[Position.NoOfCellsSouth, Position.NoOfCellsEast] := Item;
End;

Constructor Grid.Create(S, E : Integer);
Begin
  NS := S;
  WE := E;
  SetLength(CavernState, NS+1, WE+1);
End;

Procedure Grid.Reset;
```

```
Var
  Count1, Count2 : Integer;
Begin
  For Count1 := 0 To NS Do
    For Count2 := 0 To WE Do
      CavernState[Count1, Count2] := ' ';
End;

Function Game.CheckValidMove(Direction : Char) : Boolean;
Var
  ValidMove : Boolean;
Begin
  ValidMove := True;
  If Not(Direction In ['N', 'S', 'W', 'E', 'M']) Then
    ValidMove := False;
  CheckValidMove := ValidMove;
End;

Constructor Game.Create(IsATrainingGame : Boolean);
Begin
  Player := Character.Create;
  Cavern := Grid.Create(NS, WE);
  Monster := Enemy.Create;
  Flask := Item.Create;
  Trap1 := Trap.Create;
  Trap2 := Trap.Create;
  TrainingGame := IsATrainingGame;
  Randomize;
  SetUpGame;
  Play;
End;

Destructor Game.Destroy;
Begin
  Player.Free;
  Cavern.Free;
  Monster.Free;
  Flask.Free;
  Trap1.Free;
  Trap2.Free;
  Inherited;
End;

Procedure Game.DisplayLostGameMessage;
Begin
  Writeln('ARGHHHHHHH! The monster has eaten you. GAME OVER. ');
  Writeln('Maybe you will have better luck next time you play MONSTER!');
  Writeln;
End;

Procedure Game.DisplayMoveOptions;
Begin
  Writeln;
  Writeln('Enter N to move NORTH');
```

```
    Writeln('Enter E to move EAST');
    Writeln('Enter S to move SOUTH');
    Writeln('Enter W to move WEST');
    Writeln('Enter M to return to the Main Menu');
    Writeln;
End;

Procedure Game.DisplayTrapMessage;
Begin
    Writeln('Oh no! You have set off a trap. Watch out, the monster is now
awake!');
    Writeln;
End;

Procedure Game.DisplayWonGameMessage;
Begin
    Writeln('Well done! You have found the flask containing the Styxian potion.');
```

```
    Writeln('You have won the game of MONSTER!');
    Writeln;
End;

Function Game.GetMove : Char;
Var
    Move : Char;
Begin
    Readln(Move);
    Writeln;
    GetMove := Move;
End;

Function Game.GetNewRandomPosition : CellReference;
Var
    Position : CellReference;
Begin
    Repeat
        Position.NoOfCellsSouth := Random(NS+1);
        Position.NoOfCellsEast := Random(WE+1);
    Until (Position.NoOfCellsSouth > 0) Or (Position.NoOfCellsEast > 0);
    GetNewRandomPosition := Position;
End;

Procedure Game.Play;
Var
    Count : Integer;
    Eaten : Boolean;
    FlaskFound : Boolean;
    MoveDirection : Char;
    ValidMove : Boolean;
    Position : CellReference;
Begin
    Eaten := False;
    FlaskFound := False;
    Cavern.Display(Monster.GetAwake);
```

```
Repeat
  Repeat
    DisplayMoveOptions;
    MoveDirection := GetMove;
    ValidMove := CheckValidMove(MoveDirection);
  Until ValidMove;
  If MoveDirection <> 'M' Then
  Begin
    Cavern.PlaceItem(Player.GetPosition, ' ');
    Player.MakeMove(MoveDirection);
    Cavern.PlaceItem(Player.GetPosition, '*');
    Cavern.Display(Monster.GetAwake);
    FlaskFound := Player.CheckIfSameCell(Flask.GetPosition);
    If FlaskFound Then
      DisplayWonGameMessage;
    Eaten := Monster.CheckIfSameCell(Player.GetPosition);
    {This selection structure checks to see if the player has triggered
    one of the traps in the cavern}
    If (Not Monster.GetAwake) And (Not FlaskFound) And (Not Eaten) And
      ((Player.CheckIfSameCell(Trap1.GetPosition)) And
      (Not Trap1.GetTriggered)) Or
      ((Player.CheckIfSameCell(Trap2.GetPosition)) And
      (Not Trap2.GetTriggered)) Then
    Begin
      Monster.ChangeSleepStatus;
      DisplayTrapMessage;
      Cavern.Display(Monster.GetAwake);
    End;
    If (Monster.GetAwake) And (Not Eaten) And (Not FlaskFound) Then
    Begin
      Count := 0;
      Repeat
        Cavern.PlaceItem(Monster.GetPosition, ' ');
        Position := Monster.GetPosition;
        Monster.MakeMove(Player.GetPosition);
        Cavern.PlaceItem(Monster.GetPosition, 'M');
        If Monster.CheckIfSameCell(Flask.GetPosition) Then
        Begin
          Flask.SetPosition(Position);
          Cavern.PlaceItem(Position, 'F');
        End;
        Eaten := Monster.CheckIfSameCell(Player.GetPosition);
        Writeln;
        Writeln('Press enter key to continue');
        Readln;
        Cavern.Display(Monster.GetAwake);
        Count := Count + 1;
      Until (Count = 2) Or Eaten;
    End;
    If Eaten Then
      DisplayLostGameMessage;
    End;
  Until Eaten Or FlaskFound Or (MoveDirection = 'M');
End;
```

```
Function Game.SetPositionOfItem(Item : char) : CellReference;
Var
  Position: CellReference;
Begin
  Repeat
    Position := GetNewRandomPosition;
  Until Cavern.IsCellEmpty(Position);
  Cavern.PlaceItem(Position, Item);
  SetPositionOfItem := Position;
End;

Procedure Game.SetUpGame;
Var
  Position: CellReference;
Begin
  Cavern.Reset;
  If Not TrainingGame Then
  Begin
    Position.NoOfCellsEast := 0;
    Position.NoOfCellsSouth := 0;
    Player.SetPosition(Position);
    Cavern.PlaceItem(Position, '*');
    Trap1.SetPosition(SetPositionOfItem('T'));
    Trap2.SetPosition(SetPositionOfItem('T'));
    Monster.SetPosition(SetPositionOfItem('M'));
    Flask.SetPosition(SetPositionOfItem('F'));
  End
  Else
  Begin
    Position.NoOfCellsEast := 4;
    Position.NoOfCellsSouth := 2;
    Player.SetPosition(Position);
    Cavern.PlaceItem(Position, '*');
    Position.NoOfCellsEast := 6;
    Position.NoOfCellsSouth := 2;
    Trap1.SetPosition(Position);
    Cavern.PlaceItem(Position, 'T');
    Position.NoOfCellsEast := 4;
    Position.NoOfCellsSouth := 3;
    Trap2.SetPosition(Position);
    Cavern.PlaceItem(Position, 'T');
    Position.NoOfCellsEast := 4;
    Position.NoOfCellsSouth := 0;
    Monster.SetPosition(Position);
    Cavern.PlaceItem(Position, 'M');
    Position.NoOfCellsEast := 3;
    Position.NoOfCellsSouth := 1;
    Flask.SetPosition(Position);
    Cavern.PlaceItem(Position, 'F');
  End;
End;
End.
```
