2010

Gr.12 PAT 2010

EVER WINTER KNIGHTS
© 2010, Jeremy Paton, DESIGN: Jeremy Paton,. Ever Winter Knights,. www.Jesararts.net

Jeremy Paton
Trinity House
1/1/2010

Table of Contents

PROJECT SPECIFICATIONS

Pages: 4 - 6

1. Program Description:

The program that I intend to develop will be based on the popular 2D platform game style. The game revolves around movement of a character and his/her interactions with his/her surroundings; the concept is derived from Pakour, also known as free-running and has been demonstrated in the game Mirror's Edge2D. However, the freedom of movement in this game will be scaled down.

The character's objective is to progress along an environment made up of individual platform style maps. He/She will interact with objects in order to obtain items, access a location or to complete a quest. The concept is for the player to finish all the quests and thus the game story.

The character will have certain skills, such as being able to run and climb, as well as being able to battle enemies. These enemies will be controlled by an A.I. script. The player will gain levels for the amount of enemies slain, as well as each quest the player finishes. The player will also have a predefined amount of lives, which will be lost for any mistakes resulting in death. Once he has used all his lives, the active quest will be over. At the end of all quests the game will be over.

2. Program Specifications:

2.1. Input & Output:

The basic movement input methods will be the use the A, D, S and W arrow keys to move the character. The player will also be able to interact with objects by either moving around them, or by pressing an assigned key to obtain an object when the character is located over that item. The player will have an assigned key for attacking enemies.

The output method will be a combination of on screen images; these include the character itself, the active map, character statistics, enemy statistics and obtainable items. Other output methods include a form for the active quest information and a help form.

2.2. Storage and memory:

All game information will be accessed through the respective tables located in a game database.

Starting a new game will access all the information such as a player's name, level, location, starting quest and other required statistics (Life and Attack points) to begin a new game. This information will be accessed from a New Game table stored in the game database.

Continuing a previous game will access all the information including the player's name, level, location, starting quest and other required statistics (Life and Attack points) from a previous save. This information will be accessed from a Saved Game table stored in the game database. This table will be updated with the player's new statistics after each active quest is completed.

All the players' statistics will be loaded into a Character class/object at the start of a new game or a continued game. All the calculations for the characters movement and interactions will be done in this object.

Another Environment class/object will hold all the individual map classes. At present the concept is for each individual Map class to obtain its unique data, such as corresponding image, corresponding name and all interactive (on screen) object information, from a corresponding table in the game database when the player enters that specific map. Thus each individual map will have a corresponding table, holding its specific information, in the game database.

Another Opponent class/object will hold the information that an enemy will use. This class will be closely linked to the information in the Character Class and the Environment Class. Using information such as the players level an "in proportionate" enemy statistics can be created. All A.I. calculations will be calculated in this class.

The player's progress; including level, location and statistics (such as health) will be saved into respective tables in a database after each active quest is completed. Thus, by character death or exiting the game during an active quest the player will only be able to continue from the auto-save of the previously finished quest.

2.3. The GUI:

The game will be built up upon multiple graphical user interfaces. The first form the player will be presented with is the introduction form. A splash image of the game logo will appear and disappear, followed by the introduction screen itself. Here the player will have input options to begin a new game, continue from previous game, and change basic settings and exiting the game.

The next form will be the game form, this is where the player will control their character, interact with their surroundings and thus play the game. This form will contain an image that obtains its corresponding from the map classes' image field. This form will also display the characters statistics.

Other forms will include the active quest form and the help form. The active quest form will simply display information relating to the quest you are currently occupied in. This form may also include a list of previously finished quests. The help form will include basic game controls, help tips, and may include integrated demonstration videos if necessary.

2.4. Game Rules:

The game rules are quite basic the player will receive a quest at the start of the game; the player is then required to complete this quest by carrying out a certain task. During each quest a player may obtain new items which will simply increase his stats, the player may also be required to attack an enemy during a quest which will also result in his/her stats increasing. The player's stat-increases will be determined by their level or an enemy's level. When the player completes a quest their stats will increase as well as their level and score. By completing all the quests the player will finish the game.

2.4.1. Success Criteria:

- The player can complete all the quests and finish the game.
- The player can battle enemies and enemies can battle the player. ✓
- The player can only move in logical areas, steps, walkways, ladders, etc. ✓
- The player can save his progress after each successful quest and load each save. ✓
- The game successfully includes inheritance, object orientated programming, database interaction and text file interaction. ✓

2.5.Help:

The help information is stored in a text file and is loaded into the help form on show; this text is displayed in a rich edit. It explains how to play Ever Winter Knights. It includes the control scheme, item acquisition, and attacking. It also explains the games auto save and the game rules. The help form is accessed from both the menu form and the game form.

2.6.Hardware, Software and Installation Requirements:

Software Platform: Windows XP, Vista, 7. The program can be run on any 32 / 64 bit Windows system. It should be able to run as a standalone application & be accessible through a network.

Hardware: Intel Core 2 Duo and above, 160GB HDD, 512MB RAM, integrated graphics or graphics card.

Installation should take place using the standard Installshield installer. Data files and executable will be stored in the same folder – and the program will function without DLL's or other complexities – meaning that simply copying the folder to a new drive / computer will mean that the program will function there.

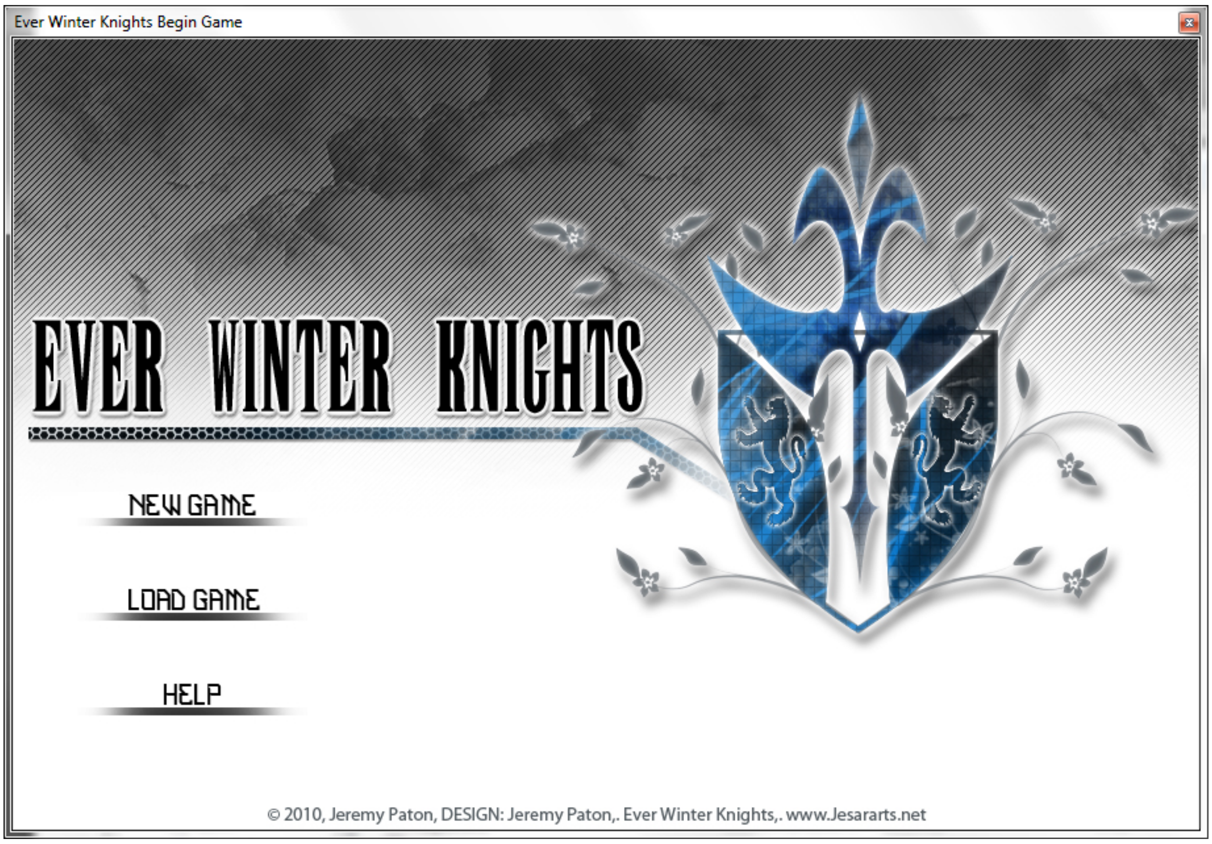
DESIGN DOCUMENTATION

Pages: 8 - 25

3. The Game Design:

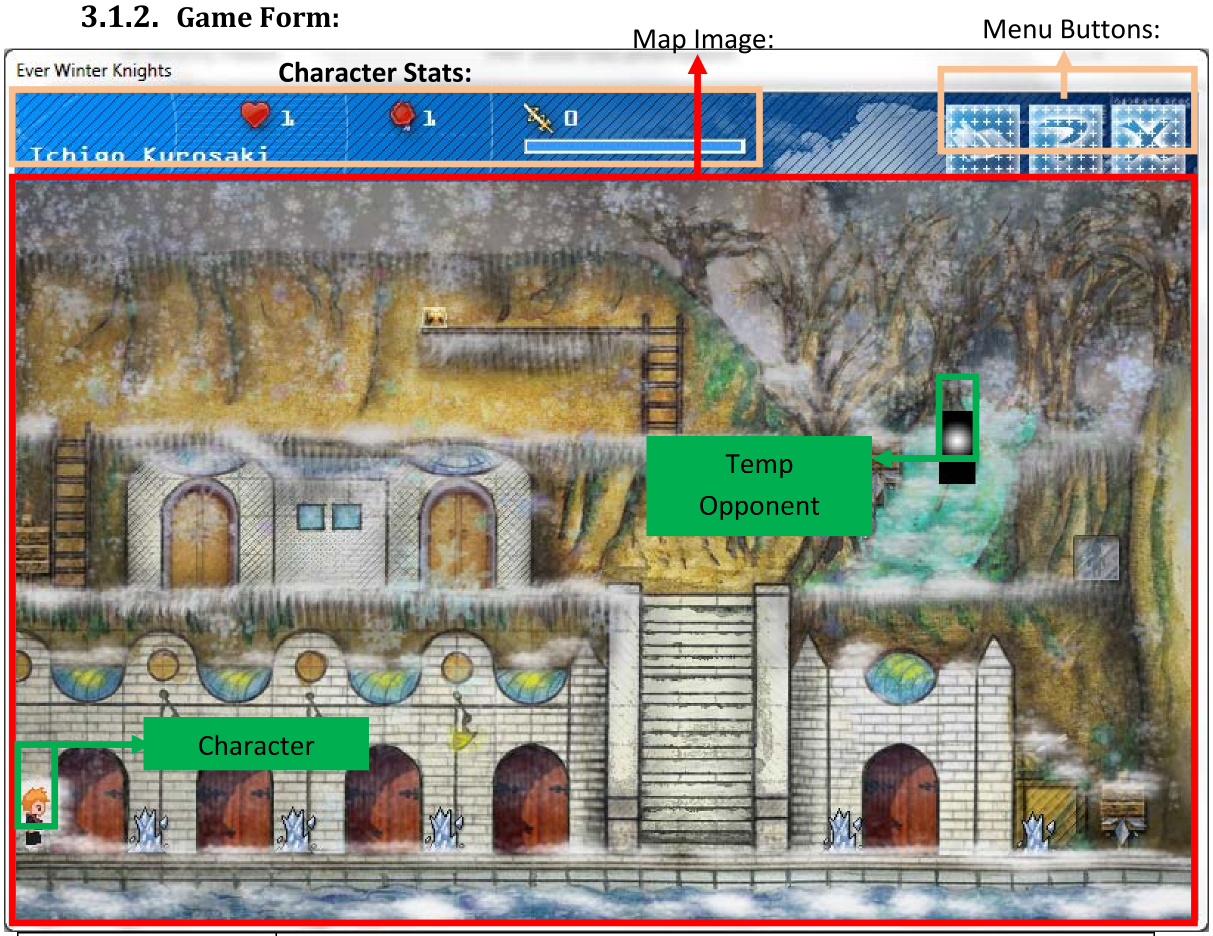
3.1. GUIs:

3.1.1. Intro Form:


Ever Winter Knights Begin Game
EVER WINTER KNIGHTS
NEW GAME
LOAD GAME
HELP
© 2010, Jeremy Paton, DESIGN: Jeremy Paton,. Ever Winter Knights,. www.Jesararts.net

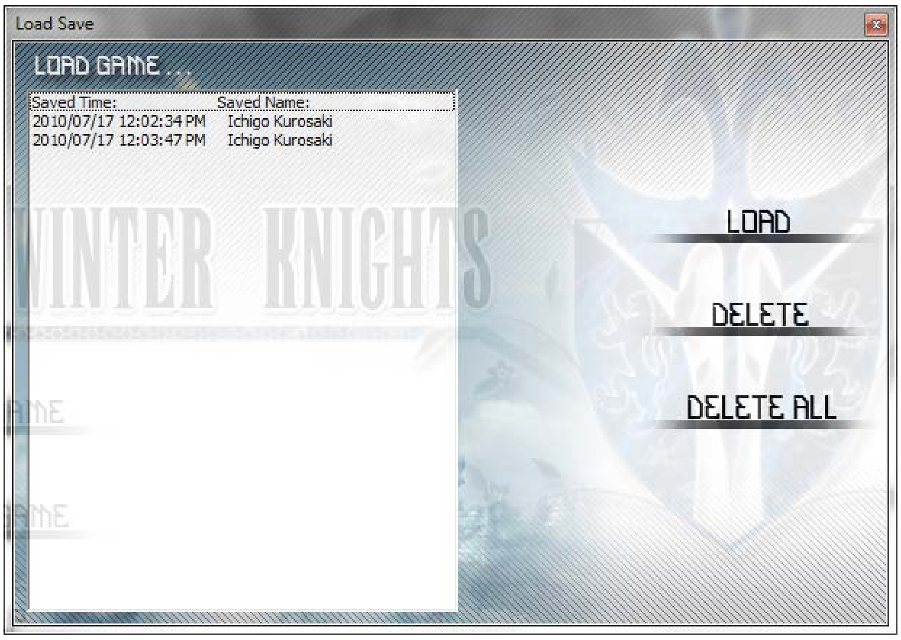
New Game:	• Starting a new game will hide the intro form and show the game form. • This will access all the information required to begin a new game. • This information will include; the auto start location, character level, character life and character attack points. • This information will be accessed from a New Game table stored in the game database.
Continue Game:	• Continuing a game will hide the intro form and show the game form. • This will access all the information from a previous save. • This information will include; the saved location, saved character level, saved character life and saved character attack points. • This information will be accessed from a Saved Game table stored in the game database.
Help	• Will show the help form.
Exit:	• This will terminate the entire application.

3.1.2. Game Form:



Map Image:	• This will hold the corresponding map image from the map classes' image field. • Map image corresponds to map character is located in. • Dynamically loaded.
Menu Buttons:	• Quest: This will display a form containing the active quest info. • Help: This will display a form containing game help info. • Exit: This will quit the game session, and display the into form.
Character Stats:	• Player Name: This is the name that the player chose at the start of the game. • Level: This displays the player's current level. • Health: This displays the number of lives the player has remaining. • Attack Points: This is the amount of damage (an integer) that the player can inflict on an enemy.
Items:	• Items will be located within each map. Their data is located in the corresponding maps class. • Player can obtain an object, which will simply increase a certain statistic.
Objects:	• These are obstacles the player must move around, upon, etc. • Their data is located in the corresponding maps class.
Character:	• This is an image of an animated .GIF • The character will be controlled by the player's inputs.

3.1.3. Load Form:



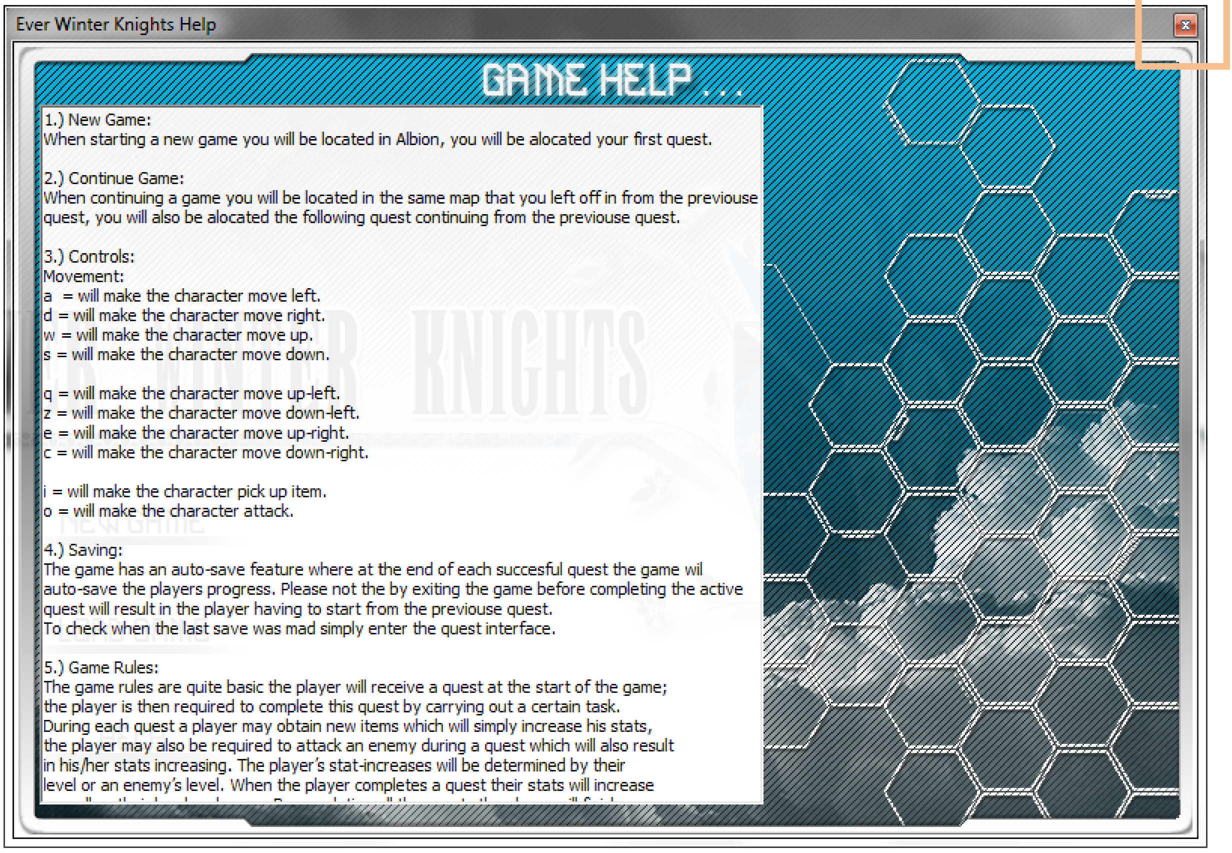
```
Saved Time:               Saved Name:
2010/07/17 12:02:34 PM    Ichigo Kurosaki
2010/07/17 12:03:47 PM    Ichigo Kurosaki
```

Load:	• Query player if they would like to load selected save. • Will initiate the load procedure, loading the selected save from the list box.
Delete:	• Query player if they would like to delete selected save. • Deletes selected save from the SavedGame table in the Game's database.
Delete All:	• Query player if they would like to delete all saves. • Deletes all saves from the SavedGame table in the Game's database.

3.2.Reports:

3.2.1. Help Form:

Menu Buttons:



1.) New Game:
When starting a new game you will be located in Albion, you will be alocated your first quest.

2.) Continue Game:
When continuing a game you will be located in the same map that you left off in from the previouse quest, you will also be alocated the following quest continuing from the previouse quest.

3.) Controls:
Movement:
a = will make the character move left.
d = will make the character move right.
w = will make the character move up.
s = will make the character move down.

q = will make the character move up-left.
z = will make the character move down-left.
e = will make the character move up-right.
c = will make the character move down-right.

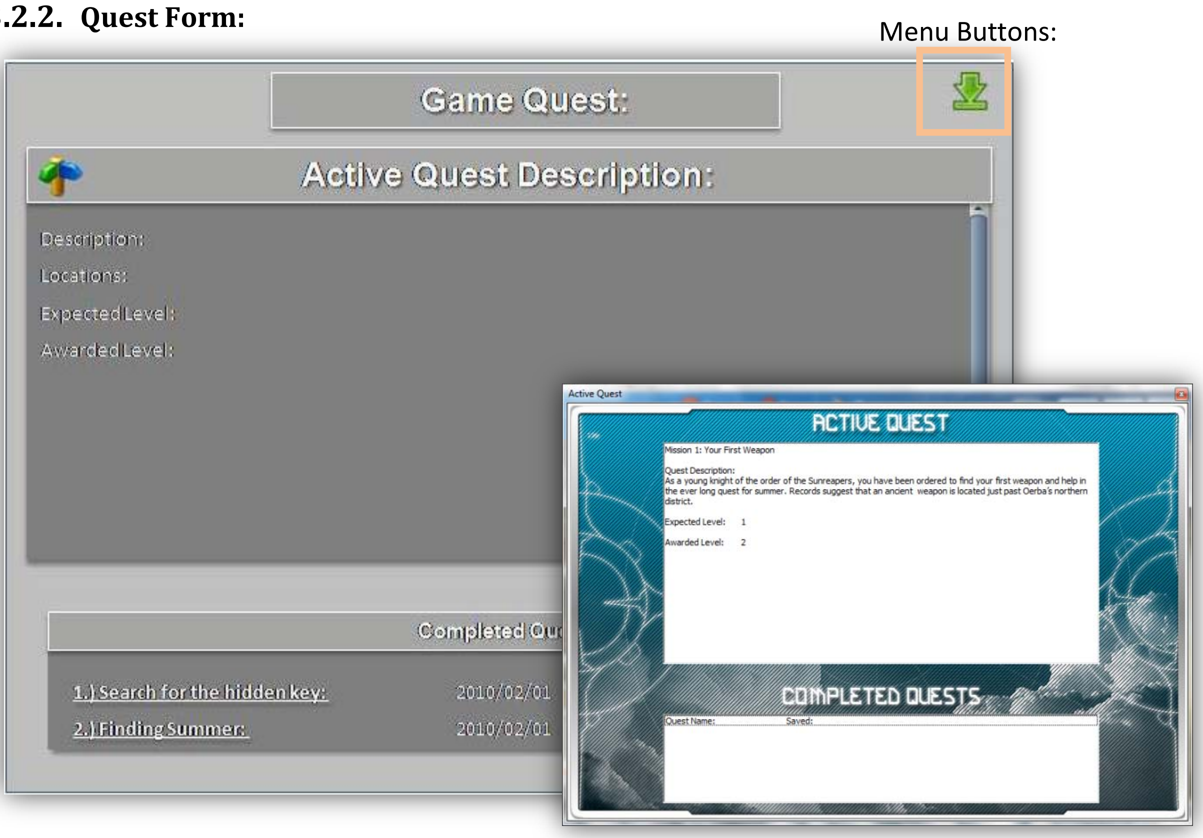
i = will make the character pick up item.
o = will make the character attack.

4.) Saving:
The game has an auto-save feature where at the end of each succesful quest the game wil auto-save the players progress. Please not the by exiting the game before completing the active quest will result in the player having to start from the previouse quest.
To check when the last save was mad simply enter the quest interface.

5.) Game Rules:
The game rules are quite basic the player will receive a quest at the start of the game; the player is then required to complete this quest by carrying out a certain task.
During each quest a player may obtain new items which will simply increase his stats, the player may also be required to attack an enemy during a quest which will also result in his/her stats increasing. The player's stat-increases will be determined by their level or an enemy's level. When the player completes a quest their stats will increase

Description Area:	• These are descriptions on how to play Ever Winter Knights. It includes control scheme, item acquisition, and attacking. It also explains the games auto save and the game rules.
Menu Buttons:	• This button will hide the report/form until the player next selects the help button from the game screen.

3.2.2. Quest Form:



1.) Search for the hidden key: 2010/02/01
2.) Finding Summer: 2010/02/01

Mission 1: Your First Weapon

Quest Description:
As a young knight of the order of the Sunreapers, you have been ordered to find your first weapon and help in the ever long quest for summer. Records suggest that an ancient weapon is located just past Oerba's northern district.

Expected Level: 1

Awarded Level: 2

Quest Name: Saved:

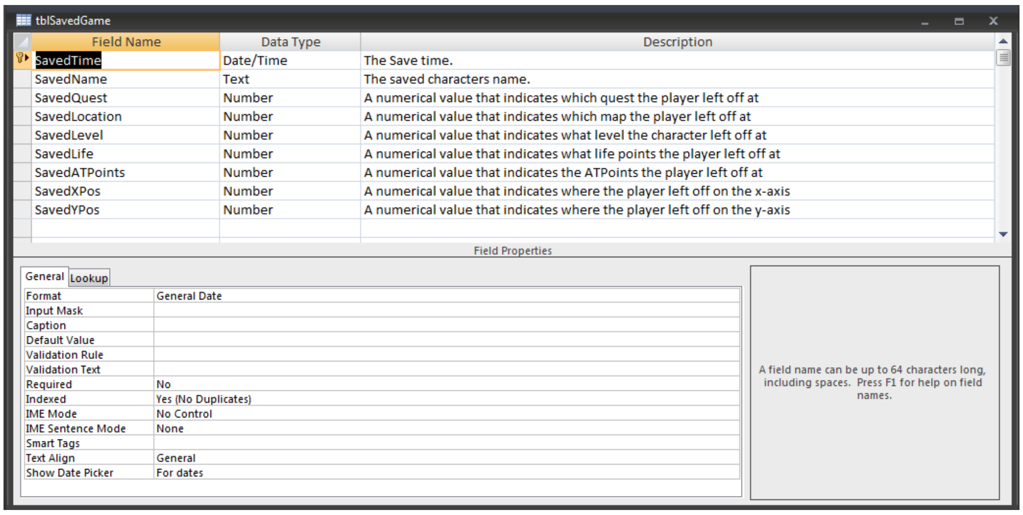
Active Quest Description:	• This region will display a description of the active/current quest. • It will also indicate the locations this quest takes place in. • It will also indicate your character's expected level and how many levels you will gain for completing the quest. • This information will be received from the Quest Table within the Game Database.
Completed Quests:	• This region will display the name of completed quests. • It will also state that there was a game save. • This information will be calculated by taking your active quest number, in the Saved Game Table, and listing all the quest names which come before the active quest. • This region will also display your game progress as a percentage.
Menu Button:	• This button will hide the report/form until the player next selects the active quest button from the game screen.

3.3. Storage and Memory:

3.3.1. Storage Design:

3.3.1.1. Saved Game Table:

This table stores all the required values to continue a saved game. These values are updated after each successful mission.



tblSavedGame

Field Name	Data Type	Description
SavedTime	Date/Time	The Save time.
SavedName	Text	The saved characters name.
SavedQuest	Number	A numerical value that indicates which quest the player left off at
SavedLocation	Number	A numerical value that indicates which map the player left off at
SavedLevel	Number	A numerical value that indicates what level the character left off at
SavedLife	Number	A numerical value that indicates what life points the player left off at
SavedATPoints	Number	A numerical value that indicates the ATPoints the player left off at
SavedXPos	Number	A numerical value that indicates where the player left off on the x-axis
SavedYPos	Number	A numerical value that indicates where the player left off on the y-axis

Field Properties

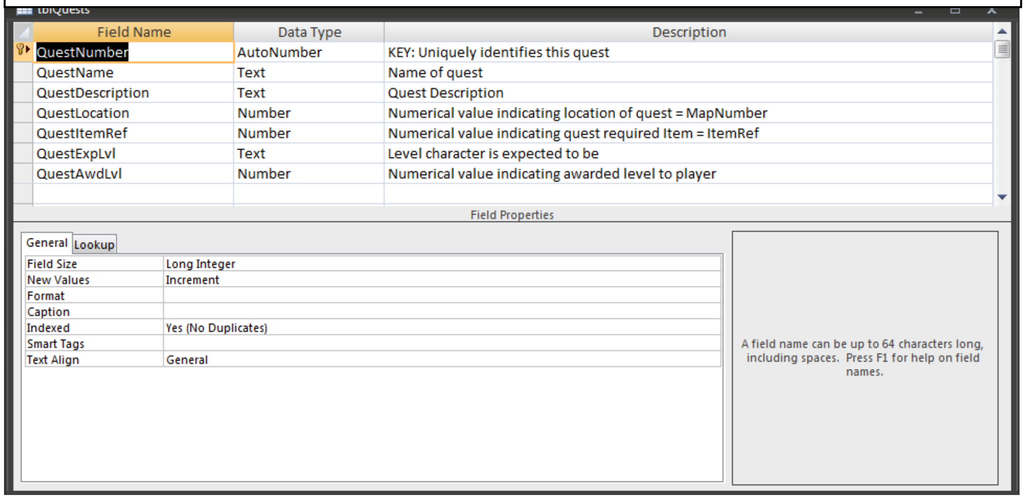
General Lookup

Format	General Date
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	No
Indexed	Yes (No Duplicates)
IME Mode	No Control
IME Sentence Mode	None
Smart Tags	
Text Align	General
Show Date Picker	For dates

A field name can be up to 64 characters long, including spaces. Press F1 for help on field names.

3.3.1.2. Quest Table:

This table stores all the values for all the playable quests of the game. This table stores fields such as the Quest Name, Quest Description, etc. The field QuestItemRef matches the ItemRef in the Items table.



tblQuests

Field Name	Data Type	Description
QuestNumber	AutoNumber	KEY: Uniquely identifies this quest
QuestName	Text	Name of quest
QuestDescription	Text	Quest Description
QuestLocation	Number	Numerical value indicating location of quest = MapNumber
QuestItemRef	Number	Numerical value indicating quest required Item = ItemRef
QuestExpLvl	Text	Level character is expected to be
QuestAwdLvl	Number	Numerical value indicating awarded level to player

Field Properties

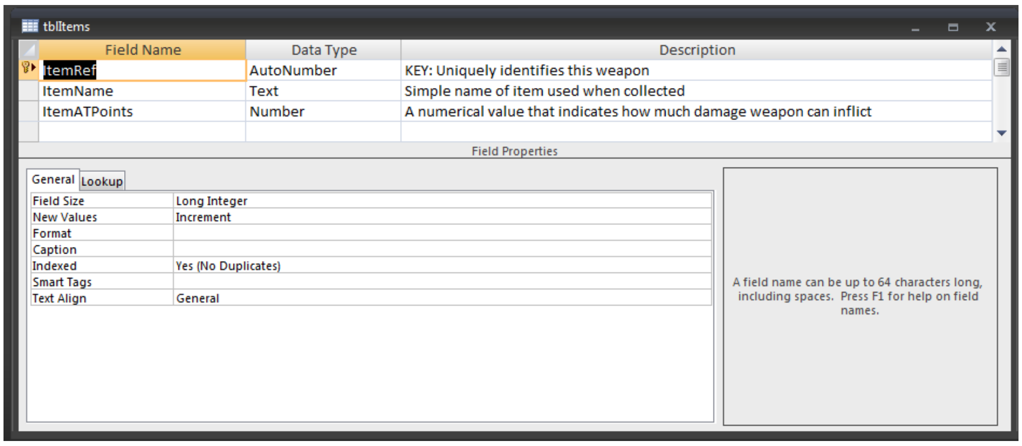
General Lookup

Field Size	Long Integer
New Values	Increment
Format	
Caption	
Indexed	Yes (No Duplicates)
Smart Tags	
Text Align	General

A field name can be up to 64 characters long, including spaces. Press F1 for help on field names.

3.3.1.3. *Items Table:*

This table stores the data associated with each item such as its ATPoints and required RefNumber. This reference number is used to identify this object when character moves over grid reference with corresponding ItemRef number


tblItems

Field Name	Data Type	Description
ItemRef	AutoNumber	KEY: Uniquely identifies this weapon
ItemName	Text	Simple name of item used when collected
ItemATPoints	Number	A numerical value that indicates how much damage weapon can inflict

Field Properties

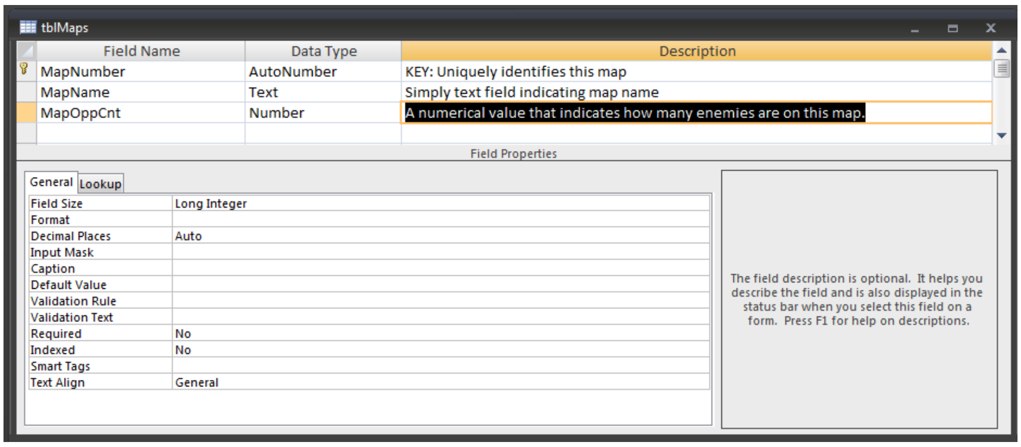
General | Lookup

Field Size	Long Integer
New Values	Increment
Format	
Caption	
Indexed	Yes (No Duplicates)
Smart Tags	
Text Align	General

A field name can be up to 64 characters long, including spaces. Press F1 for help on field names.

3.3.1.4. *Maps Table:*

This table stores all the values pertaining to each map in the game. Fields such as the MapImage dictate which JPG must be loaded when entering this location.


tblMaps

Field Name	Data Type	Description
MapNumber	AutoNumber	KEY: Uniquely identifies this map
MapName	Text	Simply text field indicating map name
MapOppCnt	Number	A numerical value that indicates how many enemies are on this map.

Field Properties

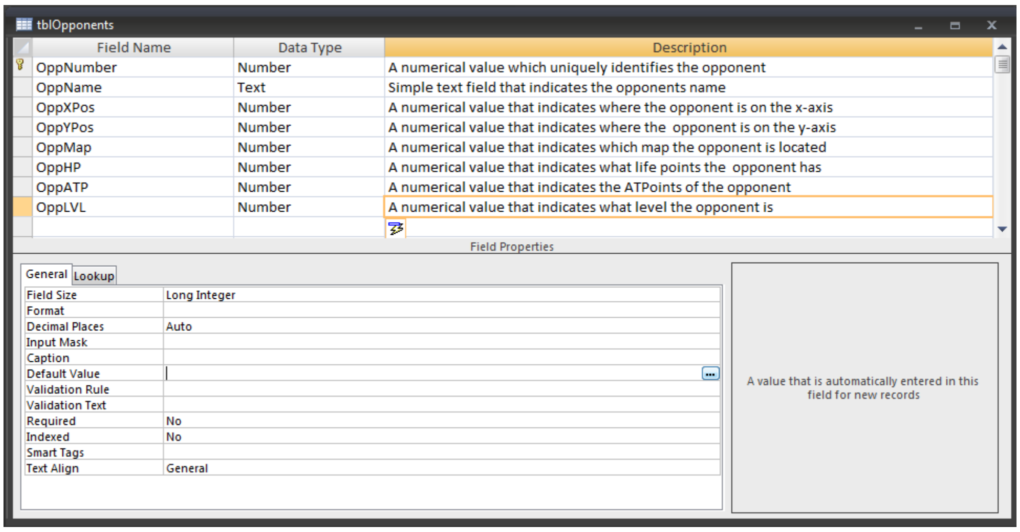
General | Lookup

Field Size	Long Integer
Format	
Decimal Places	Auto
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	No
Indexed	No
Smart Tags	
Text Align	General

The field description is optional. It helps you describe the field and is also displayed in the status bar when you select this field on a form. Press F1 for help on descriptions.

3.3.1.5. Maps Table:

This table stores all the values pertaining to each opponent in the game. Fields such as the OppLvl dictate which JPG must be loaded for an opponent and the OppMap field indicates which map that opponent is located in.



tblOpponents

Field Name	Data Type	Description
OppNumber	Number	A numerical value which uniquely identifies the opponent
OppName	Text	Simple text field that indicates the opponents name
OppXPos	Number	A numerical value that indicates where the opponent is on the x-axis
OppYPos	Number	A numerical value that indicates where the opponent is on the y-axis
OppMap	Number	A numerical value that indicates which map the opponent is located
OppHP	Number	A numerical value that indicates what life points the opponent has
OppATP	Number	A numerical value that indicates the ATPoints of the opponent
OppLVL	Number	A numerical value that indicates what level the opponent is

Field Properties

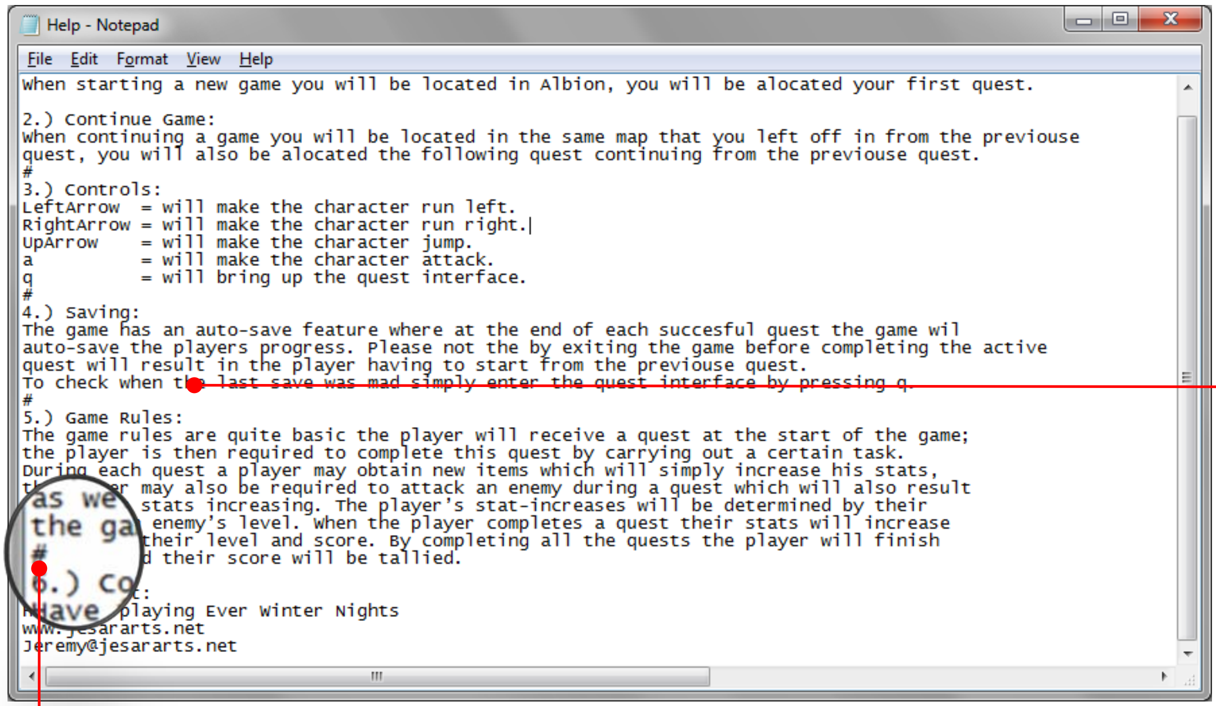
General | Lookup

Field Size	Long Integer
Format	
Decimal Places	Auto
Input Mask	
Caption	
Default Value	
Validation Rule	
Validation Text	
Required	No
Indexed	No
Smart Tags	
Text Align	General

A value that is automatically entered in this field for new records

Explanation:	• I have chosen to store the game information in the form of a database as a database is highly structured and can store large amounts of data. • The Game Database is used to hold all the information required to start a new game and continue a saved game. This database will also hold the information that relates to quests such as, what the quest entails, statistics acquired by completing the quest, etc. This database will also hold the information relating to each individual map, as well as sub tables that hold all the information relating to the objects within each individual map.

3.3.2. Storage design for help and Game Rules:



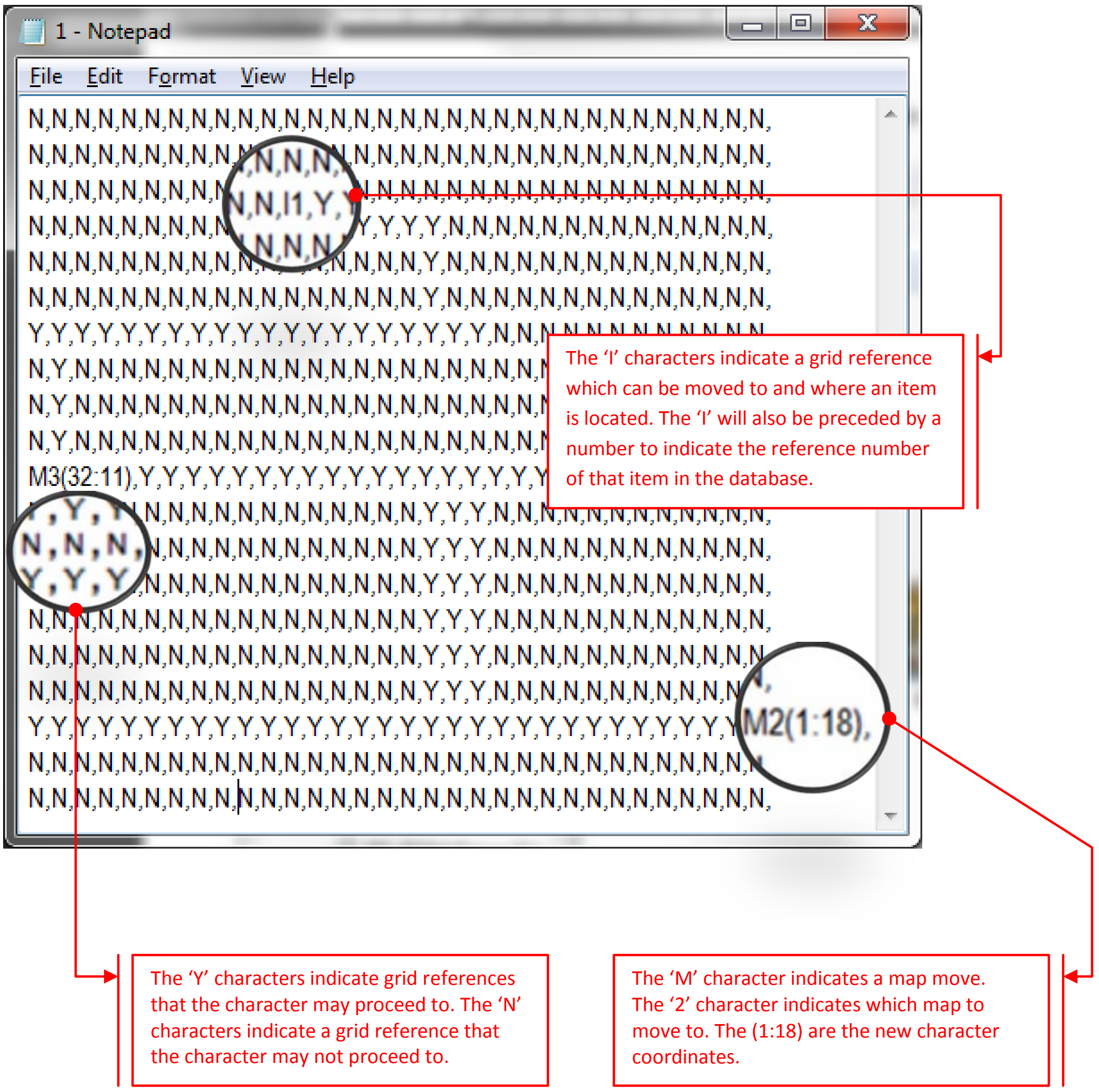
```
when starting a new game you will be located in Albion, you will be alocated your first quest.

2.) Continue Game:
when continuing a game you will be located in the same map that you left off in from the previouse
quest, you will also be alocated the following quest continuing from the previouse quest.
#
3.) Controls:
LeftArrow  = will make the character run left.
RightArrow = will make the character run right.|
UpArrow    = will make the character jump.
a          = will make the character attack.
q          = will bring up the quest interface.
#
4.) Saving:
The game has an auto-save feature where at the end of each succesful quest the game wil
auto-save the players progress. Please not the by exiting the game before completing the active
quest will result in the player having to start from the previouse quest.
To check when the last save was mad simply enter the quest interface by pressing q.
#
5.) Game Rules:
The game rules are quite basic the player will receive a quest at the start of the game;
the player is then required to complete this quest by carrying out a certain task.
During each quest a player may obtain new items which will simply increase his stats,
t[illegible]er may also be required to attack an enemy during a quest which will also result
as we[illegible] stats increasing. The player's stat-increases will be determined by their
the ga[illegible] enemy's level. When the player completes a quest their stats will increase
[illegible]their level and score. By completing all the quests the player will finish
#[illegible]d their score will be tallied.

6.) Co[illegible]t:
Have [illegible]playing Ever Winter Nights
www.jesararts.net
Jeremy@jesararts.net
```

The '#' symbols are used in the structure of the help. Each '#' symbol represents the beginning of a new help section. Each section is then placed in the corresponding video description's text box.

Each new section consists of a heading and a subsequent description.

Explanation:	• I have chosen to store the help text in a text file has the descriptions can range over 255 words, making it impractical to try store in a database. The text file is more suited to holding paragraphs of text. • The videos used within the help form will be stored as .AVIs and will be located within a folder within the game folder. These videos will be loaded into the help form in their corresponding position. The videos will be certain screen captured videos while I play the game. • The description of each video will be stored either in a text file or within the database. This text will be loaded into the game at the same time the videos are and will be displayed on the help form alongside the videos. • Along with the basic help outlined in this report will be the basic game rules. The game rules are a basic summary of what the player is requested to achieve in order to finish the game.

3.3.3. Storage design for single map:



```
N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,
N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,
N,N,N,N,N,N,N,N,N,N,N,I1,Y,Y,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,
N,N,N,N,N,N,N,N,N,N,N,N,Y,Y,Y,Y,N,N,N,N,N,N,N,N,N,N,N,N,N,N,
N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,Y,N,N,N,N,N,N,N,N,N,N,N,N,N,N,
N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,Y,N,N,N,N,N,N,N,N,N,N,N,N,N,N,
Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,N,N,N,N,N,N,N,N,N,N,N,N,
N,Y,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,
N,Y,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,
N,Y,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,
M3(32:11),Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,
,Y,Y,N,N,N,N,N,N,N,N,N,N,N,N,Y,Y,Y,N,N,N,N,N,N,N,N,N,N,N,N,
N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,Y,Y,Y,N,N,N,N,N,N,N,N,N,N,N,N,
Y,Y,Y,N,N,N,N,N,N,N,N,N,N,N,N,Y,Y,Y,N,N,N,N,N,N,N,N,N,N,N,N,
N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,Y,Y,Y,N,N,N,N,N,N,N,N,N,N,N,N,
N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,Y,Y,Y,N,N,N,N,N,N,N,N,N,N,N,N,
N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,Y,Y,Y,N,N,N,N,N,N,N,N,N,N,N,N,
Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,Y,M2(1:18),
N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,
N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,
```

The 'I' characters indicate a grid reference which can be moved to and where an item is located. The 'I' will also be preceded by a number to indicate the reference number of that item in the database.

The 'Y' characters indicate grid references that the character may proceed to. The 'N' characters indicate a grid reference that the character may not proceed to.

The 'M' character indicates a map move. The '2' character indicates which map to move to. The (1:18) are the new character coordinates.

Explanation:	• I have chosen to store each map's object locations in a text file as it is a simple approach, is easy to create and alter and because it does not create repetitive fields in a database. The text file is more suited to holding larger amounts of text. • Each character will be extracted into the corresponding map's 2D object array. • Each map will have a corresponding map-object text file. The map number field indicates what map-object text field must be used and what image applies to that map.

3.4.Objects and Classes:

TMap Object	
Fields	
Field	**Description**
ObjArr: Array (String);	2D array that stores grid reference for each map. Stores the values indicating ability to move to next grid reference or inability
SizeX: Integer;	A numerical value that stores the "size" of the array on the X axis.
SizeY: Integer;	A numerical value that stores the "size" of the array on the Y axis.
MapName: String;	Used to identify which map player is located at
Methods	
Method	**Description**
Constructor **Create** (MN: Integer; MNa:String);	Instantiates the object with the Map Number & Map Name data passed as parameters
Function **GetMapName:** String;	Returns the Map Name
Procedure **SetMapName** (MNa: String);	Changes the Map Name
Procedure **CalcVirtualGrid;**	Builds up the 2D array from a corresponding text file
Procedure**Extract** (FLine: String);	Extracts each value from text file to be inserted into the 2D array. Used in conjunction with CalcVirtualGrid
Function **ClacCollision** (FX,FY: Integer): Boolean;	Used to calculate whether a collision will occur for next movement. Returns a true or false value. Determined by whether a collision will occur or not
Function **CalcCollision**(FX,FY: Integer): Boolean;	Used to calculate whether a collision will occur for next diagonal movement.
Function **CalcMovment** (Direction: String): Boolean;	Used to calculate the characters movement in conjunction with the CalcCollision
Function **CalcDiagMovment**(Direction: Char): Boolean;	Used to calculate the characters movement in the diagonal in conjunction with the CalcDiagCollision
Procedure **CalcVirtualGrid**;	Used to populate the 2D array with the corresponding text file values.
Procedure **CalcMapMove** (New:String);	Used when player moves between maps. Re-populates the array, sets new character coordinates and loads new opponents.
Function **CalcIsQuestItem**(FRef: String): Boolean;	Used to check if obtained item is a quest item and if so, calls save game procedure and changes active quest.
Function **CalcItem**: Boolean;	Used when obtaining an item.

Explanation:	• The Environment class will hold an array of Map Classes. The CalcMovement method calculates movements, by adjusting the character positions (in their object) in conjunction with checking for collisions through the CalcCollision method. • The Environment Class will also hold the CalcCollision method which determines whether a character or opponent will collide into an object, or is situated next to an obtainable item. • The Individual Map Class will store all information describing the map, such as the Map Number and Map Name. This class will also contain a 2D array which is the structure of each map. The 2D array will hold a 'Y' value indicating a movable position, a 'N' value indicating a non-movable position and a 'I+(ItemRefNo) indicating a movable position with an item and which item it is.

TEntity Object	
Fields	
Private Fields	**Description**
OppArr: Array (TBase);	Array that stores the (Inherited) opponent object(s).
Size: Integer;	A numerical value that stores the "size" of the array.
Index: Integer;	A numerical value that indicates which opponent in battle.
Methods	
Method	**Description**
Constructor Create;	Instantiates the object.
Procedure(s)/Function(s) [**A.I**] • Procedure **CalcCharAction;** • Procedure **CalcOppAction;** • Function **CalcAttack: Boolean;** • Procedure **CalcOppDeath;**	**All calculations needed to control how an enemy behaves, e.g. when to attack, etc.**
Procedure **CalcOpponentArray;**	Used to populate the Opponent array with the corresponding opponent's values.
Procedure **CalcGameWin;**	Used to check if game is won, returns to intro form. Has the player depleted his life and lost the active quest.
Procedure **CalcGameLoss;**	Used to check if game is Lost, returns to intro form.
Procedure **SaveGame;**	Complex saving method called after each successful quest.

TBase Object **(Inheritor Object)**	
Fields	
Protected Fields	**Description**
EntityLevel: Integer ;	Indicates what level entity is at. **Also used to calculate (in proportion) enemy stats.**
EntityLife: Integer;	Indicates the life points entity is allocated
EntityATPoints: Integer ;	Indicates how much damage entity can inflict. (CharATP = ItemATP) avoids need to check if player already obtained an item as it simply won't change players ATP.
EntityMap: Integer;	Is used in the loading of a corresponding map object. Entities current lactation.
EntityXPos: Integer;	Used to store where the entity is located on the X-axis.
EntityYPos: *Integer;*	Used to store where the entity is located on the Y-axis.
Methods	
Method	**Description**
Constructor Create (EL, Ehp, Eat,EM, Ex, Ey: Integer)	Instantiates the object with all the required fields data passed as parameters
Function **Get(*FieldName*):** *Type;*	Returns the field value
Procedure **Set(*FieldName*)** *(Parameter: Type);*	Changes the field value

TCharacter (Inherited Object)	
Fields	
Field	**Description**
All Inherited Fields	
***SaveID:** String;*	Complex ID used in the saving of the game to match save and current game session.
CharName: String;	Indicates to player his characters name, selected at beginning of game.
CharQuest: Integer ;	Indicates active quest.
Methods	
Method	**Description**
Constructor Create (EL, Ehp, Eat,EM, Ex, Ey, CQ: Integer ; CN: Sting)	Instantiates the object with all the required fields data passed as parameters
Function **Get(*FieldName*):** *Type;*	Returns the field value
Procedure **Set(*FieldName*)** *(Parameter: Type);*	Changes the field value

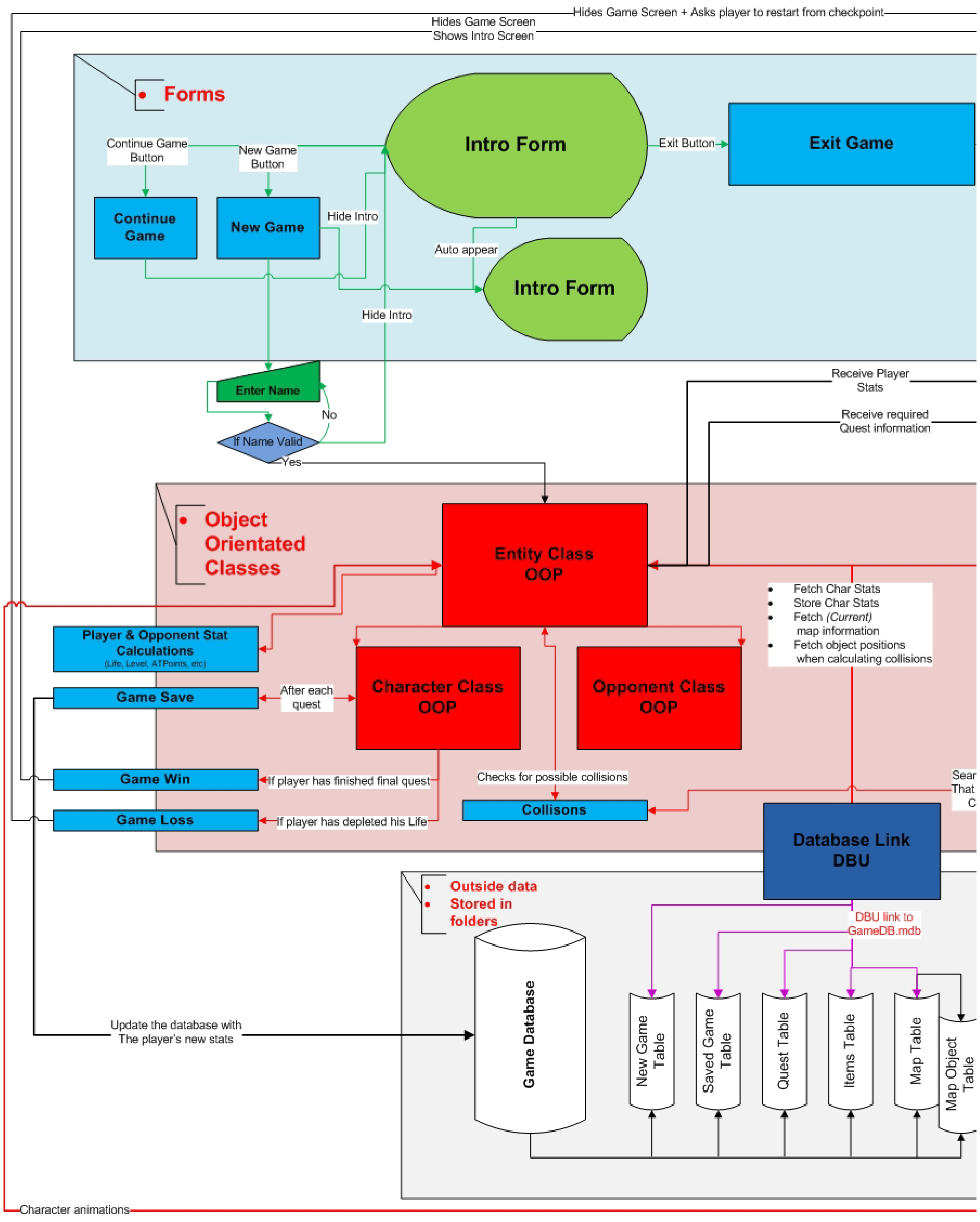
Topponent Object (Inherited Object)	
Fields	
Field	**Description**
All Inherited Fields	
XPos, YPos: Integer;	Numerical value indicating the pixel position of opponent on the X axis and Y axis.
Enemy: TBitmap;	Field used to hold each opponents image determined by its level.
Methods	
Method	**Description**
Constructor Create (Lvl, HP, ATP, M, XP, YP: Integer);	Instantiates the inherited object with all the required fields, data passed as parameters

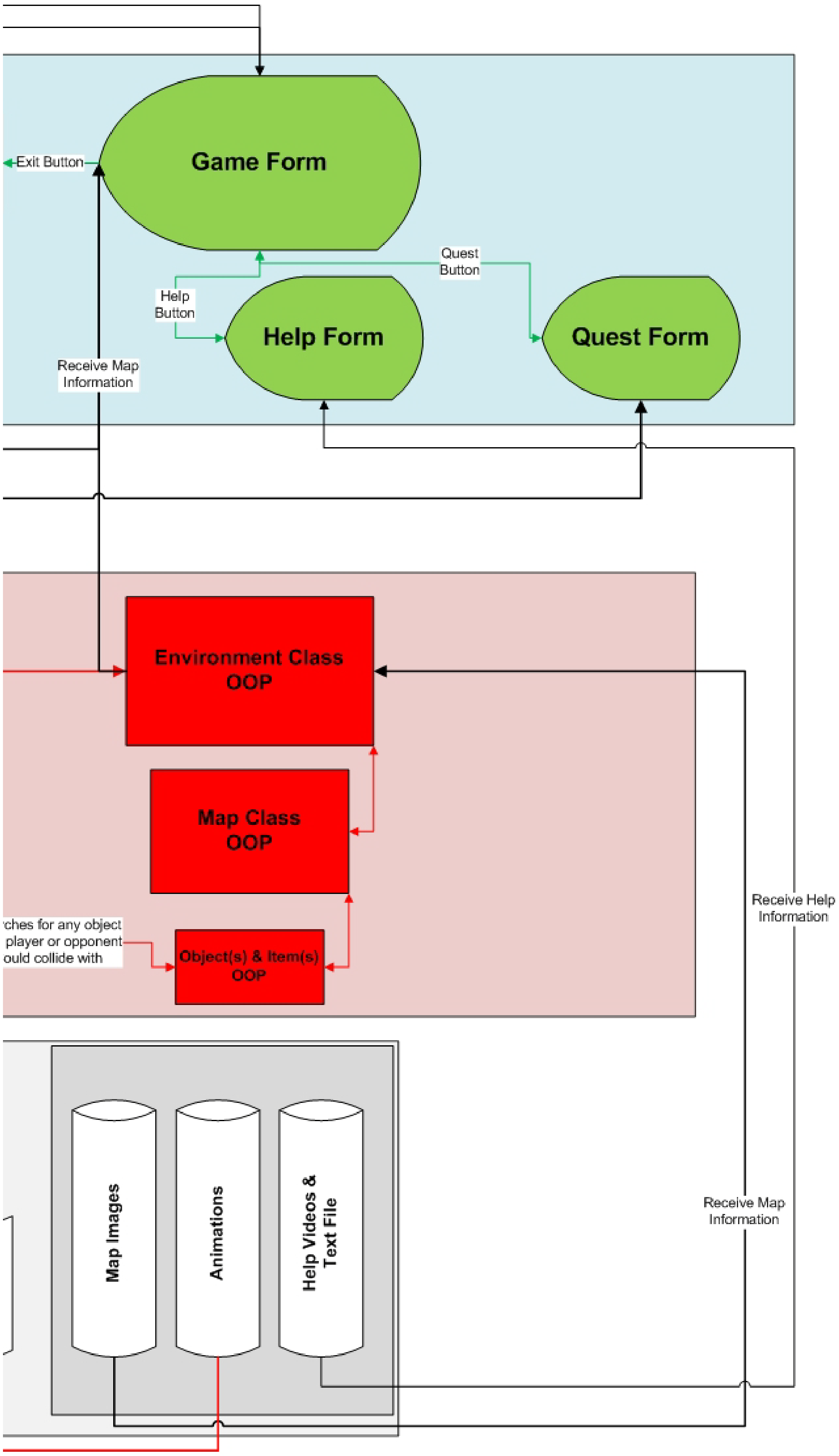
Explanation: Entity:	• Calculations such as the entities level, attack points, lives will be processed in this class. • This class will also store the A.I methods, which will be all the calculations that control many elements of an opponent. These elements include when searching for when the character is 1 position away and when to attack.

Explanation: TOpponent Object	• The Enemy object will hold all the information describing a single enemy, within fields. This information is very similar to that of the characters information. • The Enemy object will also hold all the methods that are used to calculate a single enemies Level, ATPoints, Lives.

Explanation: TCharacter Object	• This is the Character Class which will be used to store all information describing the character, in fields, such as the characters; level, lives, attack points, and location. The character position will consist of two x & y coordinates. • This information will be collected from default values, if the player has selected a new game, or will be collected from the Saved Game Table, if the player has selected to continue a game. The information contained in this class will be saved in the Game Save Table within the Game Database.

3.5. Dataflow Diagram:




Exit Button
Game Form
Quest Button
Help Button
Help Form
Quest Form
Receive Map Information
Environment Class OOP
Map Class OOP
ches for any object player or opponent ould collide with
Object(s) & Item(s) OOP
Receive Help Information
Map Images
Animations
Help Videos & Text File
Receive Map Information

3.6. Critical Algorithms:

🟩	>> Comments
🟦	>> SQL

The following algorithms are considered to be the most critical in my program as they are the core methods used in the game. These methods are used by multiple events repeatedly thought out the running of the game. Generally they are the algorithms that take the longest to code due to the level of logic, maths and order of each line of code.

1. *Game Save:*

```
  SavedTime« DateToStr(Date)+' '+TimeToStr(Time)

  CheckSQL« 'SELECT SavedTime FROM tblSavedGame WHERE SavedTime Like
"'+TheCharacter.GetSaveID+'"'

  TempID« (Query(CheckSQL))

  //Checks to see if player has previous save
  If TempID = TheCharacter.GetSaveID
  Then //Will only update their previous save
    SaveSQL« 'UPDATE tblSavedGame SET ' +
    'SavedTime = "'+SavedTime+'" ,' +
    'SavedName = "'+TheCharacter.GetNaam+'" ,' +
    'SavedQuest = "'+inttostr(TheCharacter.GetQuest)+'" ,' +
    'SavedLocation = "'+inttostr(TheCharacter.GetMap)+'" ,' +
    'SavedLevel = "'+inttostr(TheCharacter.GetLevel)+'" ,' +
    'SavedLife = "'+inttostr(TheCharacter.GetLife)+'" ,' +
    'SavedATPoints = "'+inttostr(TheCharacter.GetATPoints)+'" ,' +
    'SavedXPos = "'+inttostr(TheCharacter.GetXpos)+'" ,' +
    'SavedYPos = "'+inttostr(TheCharacter.GetYpos)+'" WHERE SavedTime
LIKE "'+TheCharacter.GetSaveID+'"'

    Change(SaveSQL)

  Else
      SaveSQL« 'INSERT INTO tblSavedGame
VALUES("'+SavedTime+'","'+TheCharacter.GetNaam+'","'+inttostr(TheChara
cter.GetQuest)+'","'+inttostr(TheCharacter.GetMap)+'","'+inttostr(TheC
haracter.GetLevel)+'","'+inttostr(TheCharacter.GetLife)+'","'+inttostr
(TheCharacter.GetATPoints)+'","'+inttostr(TheCharacter.GetXPos)+'","'+
inttostr(TheCharacter.GetYPos)+'")'
      Change(SaveSQL)
    end
    Show message 'Game Saved!'
```

2. Procedure CalcVirtualGrid:

```
  SizeY« 0;
  Assign File (RamFile,'tdMaps/'+(integer to string (TheCharacter (call)
GetMap)+'.txt'))

  If File Exists ('tdMaps/'+(integer to string (TheCharacter (call)
GetMap))+'.txt')

    Then Reset (RamFile)
    Else Rewrite (RamFile)
  Reset (RamFile)
  (Repeat)(While) not End Of File (RamFile)

      Read Line (From RamFile to Line)
      Increase SizeY;
      Extract (Line);

  Close File (RamFile)
```

3. Procedure Extract:

```
  SizeX« 0;
  Posi« position of (',') in FLine
    (Repeat)(While) Length of (FLine) > 0 do
        Increase (SizeX)
        Object Array [SizeX,SizeY]« Copy From (FLine) by 1 to  (Posi-1)
        //Builds up the 2D array from a corresponding text file
        Delete (FLine) by 1 to (posi)
```

4. Procedure Calc Movment:

```
    XPos« TheCharacter (call) GetXPos
    YPos« TheCharacter (call) GetYPos
    If Direction = 'W'
      Then Ypos« Ypos-1
      Else If Direction = 'S'
        Then Ypos« Ypos+1
        Else If Direction = 'D'
          Then Xpos« Xpos+1
          Else Xpos« Xpos-1;
    If CalcCollision(XPos,YPos) = True //Calls collision checker.
      Then
        TheCharacter (call) SetXPos(Xpos)
        TheCharacter (call) SetYPos(Ypos)
        If Uppercase (Object Array [XPos,YPos][1]) = 'M'
          Then CalcMapMove(Object Array [XPos,YPos])
        Result« True //On screen movement can take place.

      Else Result« False
```

5. *Procedure Calc Collision:*

```
If (Fx >= 1) AND (Fx <= (SizeX))
  Then If (FY >= 1) AND (FY <= (SizeY))
    Then If Uppercase(Object Array [Fx,Fy]) <> 'N'
    Then If (TheEntity (call) CalcOppCollision(Fx,Fy) = False)
      Then Result« True
      Else Result« False
```

6. *Procedure Map Move:*

```
CopyMapLeng« position of ('(') In (New)
NewMapNumber« string to integer Copy (New) By 2 to (CopyMapLeng-2)

Delete (New) by 1 to (CopyMapLeng-1)

CopyXleng« position of (':') in (New)
NewXPos« string to integer Copy (New) by 2 to (CopyXleng-2)

CopyYleng« Length of ((New)-CopyXleng)
NewYPos« string to integer Copy (New) by (CopyXleng+1) to (CopyYleng-1)

TheCharacter (call) SetMap(NewMapNumber)
TheCharacter (call) SetXPos(NewXPos)
TheCharacter (call) SetYPos(NewYPos)
CalcVirtualGrid;

With Game Form

    For Loop« 1 to cnt do
        Img Array [loop] Destroy //Clear array of opponent images.

    Cnt« 0;
    TimOppAI.Enabled« False //While re-populating Entity.
    TheEntity (call) CalcOpponentArray
    //Should call entity create and new opponent for new map.

FrmGameUI.TimOppAI.Enabled« True
//Re-enables running of opponent procedures.
```

TECHNICAL DOCUMENTATION

Pages: 27 - 64

4. Test Plans & Testing:

INPUT FROM USER

New Game:				
1.) Enter Name				
Data	**Data E.G.**	**Expected Results**	**Action Required**	**Actual Result**
Normal	Jeremy	Check length is appropriate	User Input	**Accepted**
Extreme Value	*‘ Blank ‘*	Renter name message	User Re-Input	**Rejected**
Erroneous Data	7 , % ,)	Check length is appropriate	User Input	**Accepted**
Load Game:				
1.) Load Game				
Data	**Data E.G.**	**Expected Results**	**Action Required**	**Actual Result**
Normal	Click on Button	Load the save selected in the list box	User Input	**Accepted**
Erroneous Data	Click around button	Do nothing until user clicks ON button	N/A	**No Action**
2.) Delete Save				
Data	**Data E.G.**	**Expected Results**	**Action Required**	**Actual Result**
Normal	Click on Button	Delete the save selected in the list box	User required to click yes or no to confirm	**Accepted**
Erroneous Data	Click around button	Do nothing until user clicks ON button	N/A	**No Action**
3.) Delete All Saves				
Data	**Data E.G.**	**Expected Results**	**Action Required**	**Actual Result**
Normal	Click on Button	Delete all saves independent on selection	User required to click yes or no to confirm	**Accepted**
Erroneous Data	Click around button	Do nothing until user clicks ON button	N/A	**No Action**
Playing Game:				
1.) Movment				
Data	**Data E.G.**	**Expected Results**	**Action Required**	**Actual Result**
Normal	'a','w','s','d' ‘z’,’q’,’e’,’c’	Character should move in specific direction	User Input	**Accepted**
Extreme Value	‘A','W','S','D' ‘Z’,’Q’,’E’,’C’	Character should move in specific direction	User Input	**Accepted**
Erroneous Data	7 , % ,)	Ignore key press	N/A	**No Action**
2.) Acquire Item/Attack				
Data	**Data E.G.**	**Expected Results**	**Action Required**	**Actual Result**
Normal	‘i’,’o’	Character should acquire item or attack	User Input	**Accepted**
Extreme Value	‘I’,’O’	Character should acquire item or attack	User Input	**Accepted**
Erroneous Data	7 , % ,)	Ignore key press	N/A	**No Action**

5. The Code:

5.1. MenuUI:

```
**unit MenuUI;**

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, jpeg, ExtCtrls, CharacterU, LoadUI, HelpUI, GameUI, EntityU, MapU, DBU, MMSystem;

type
  TFrmMenuUI = class(TForm)
    ImgMenu: TImage;
    ImgNewGame: TImage;
    ImgLoadGame: TImage;
    ImgHelp: TImage;
    FadeTimer: TTimer;
    procedure ImgNewGameMouseEnter(Sender: TObject);
    procedure ImgNewGameMouseLeave(Sender: TObject);
    procedure ImgLoadGameMouseEnter(Sender: TObject);
    procedure ImgLoadGameMouseLeave(Sender: TObject);
    procedure ImgHelpMouseEnter(Sender: TObject);
    procedure ImgHelpMouseLeave(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FadeTimerTimer(Sender: TObject);
    procedure ImgNewGameClick(Sender: TObject);
    procedure ImgLoadGameClick(Sender: TObject);
    procedure ImgHelpClick(Sender: TObject);
    procedure FormShow(Sender: TObject);
  private
    { Private declarations }
  public
    **Procedure BuildGame(var IsNew: Boolean; WhSave: String);**
  end;

var
  FrmMenuUI: TFrmMenuUI;

implementation

{$R *.dfm}
```

```
//The method used to load all forms, call all constructor methods
//when starting a new game or loaded one.
procedure TFrmMenuUI.BuildGame(var IsNew: Boolean; WhSave: String);
var
 Naam,SaveID , NaamSQL, MapNaam, MapNaamSQL: String;
 Quest, Level, Life, ATPoints, Map, XPos, YPos: Integer;
 QuestSQl, LevelSQL, LifeSQL, ATPointsSQL, MapSQL, XPosSQL, YPosSQL: String;

begin

 If IsNew = True //Checks if this is a new game or a loaded game.
  Then Begin
   Naam:= Inputbox('Please Enter:','Your Character Name','Ichigo Kurosaki');
   While Length(Naam) > 20 do
    Begin
     Naam:= Inputbox('Please Enter:','Your Character Name'+#13+'Name can not be longer than 20
characters','Sherlock Holmes');
    End;
   SaveID:= DateToStr(Date)+' '+TimeToStr(Time);
   Quest:= 1;
   Level:= 1;
   Life:= 1;
   ATPoints:= 0;
   Map:= 1;
   XPos:= 1; //Determined by the starting location of starting map.
   YPos:= 18; //Determined by the starting location of starting map.
  End
  Else begin
   SaveID:= WhSave;

   NaamSQL:= 'SELECT SavedName FROM tblSavedGame WHERE SavedTime LIKE '''+WhSave+'''';
   Naam:=  (Query(NaamSQL));

   NaamSQL:= 'SELECT SavedName FROM tblSavedGame WHERE SavedTime LIKE '''+WhSave+'''';
   Naam:=  (Query(NaamSQL));

   QuestSQL:= 'SELECT SavedQuest FROM tblSavedGame WHERE SavedTime LIKE '''+WhSave+'''';
   Quest:= strtoint((Query(QuestSQL)));

   LevelSQL:= 'SELECT SavedLevel FROM tblSavedGame WHERE SavedTime LIKE '''+WhSave+'''';
   Level:= strtoint((Query(LevelSQL)));

   LifeSQL:= 'SELECT SavedLife FROM tblSavedGame WHERE SavedTime LIKE '''+WhSave+'''';
   Life:=  strtoint((Query(LifeSQL)));

   ATPointsSQL:= 'SELECT SavedATPoints FROM tblSavedGame WHERE SavedTime LIKE '''+WhSave+'''';
   ATPoints:=  strtoint((Query(ATPointsSQL)));

   MapSQL:= 'SELECT SavedLocation FROM tblSavedGame WHERE SavedTime LIKE '''+WhSave+'''';
   Map:= strtoint((Query(MapSQL)));

   XPosSQL:= 'SELECT SavedXPos FROM tblSavedGame WHERE SavedTime LIKE '''+WhSave+'''';
   XPos:=  strtoint((Query(XPosSQL)));

   YPosSQL:= 'SELECT SavedYPos FROM tblSavedGame WHERE SavedTime LIKE '''+WhSave+'''';
   YPos:=  strtoint((Query(YPosSQL)));
  end;
 FrmMenuUI.Hide;
 sndPlaySound(nil, SND_ASYNC or SND_LOOP);
 FrmGameUI.Show;
 TheCharacter:= TCharacter.Create(Level, Life, ATPoints, Map, XPos, YPos, Quest, Naam, SaveID);
 TheEntity:= TEntity.Create;
 TheEntity.CalcOpponentArray;

 //Neccissary to always be loaded,  independent on new game or load game.
 MapNaamSQL:= 'SELECT MapName FROM tblMaps WHERE MapNumber LIKE "'+inttostr(Map)+'"';
 MapNaam:= (Query(MapNaamSQL));

 TheMap:= TMap.Create(MapNaam);

 //Setts up form.
 TheMap.CalcVirtualGrid; //Used to populate the virtual grid.
 FrmGameUI.UpdateForm;
end;

procedure TFrmMenuUI.FormCreate(Sender: TObject);
begin
 OpenDB;
end;

//Play music on form create
procedure TFrmMenuUI.FormShow(Sender: TObject);
begin
 sndPlaySound('tdGame\music\EverWinterKnights.wav', SND_ASYNC or SND_LOOP);
end;

//Start a new game
procedure TFrmMenuUI.ImgNewGameClick(Sender: TObject);
var
 isNew: Boolean;
begin
 isNew:= True;
 BuildGame(isNew,'');
end;

//Load game form
procedure TFrmMenuUI.ImgLoadGameClick(Sender: TObject);
begin
 FrmLoadUI.Show;
end;

//Menu option mouse effects
procedure TFrmMenuUI.ImgLoadGameMouseEnter(Sender: TObject);
begin
 imgLoadGame.Picture.LoadFromFile('tdGame\images\buttons\Load_Game_Over.jpg');
end;

procedure TFrmMenuUI.ImgLoadGameMouseLeave(Sender: TObject);
begin
 imgLoadGame.Picture.LoadFromFile('tdGame\images\buttons\Load_Game.jpg');
end;
```

```
procedure TFrmMenuUI.ImgNewGameMouseEnter(Sender: TObject);
begin
 imgNewGame.Picture.LoadFromFile('tdGame\images\buttons\New_Game_Over.jpg');
end;

procedure TFrmMenuUI.ImgNewGameMouseLeave(Sender: TObject);
begin
 imgNewGame.Picture.LoadFromFile('tdGame\images\buttons\New_Game.jpg');
end;

procedure TFrmMenuUI.ImgHelpClick(Sender: TObject);
begin
 FrmHelpUI.Show;
end;

procedure TFrmMenuUI.ImgHelpMouseEnter(Sender: TObject);
begin
 imgHelp.Picture.LoadFromFile('tdGame\images\buttons\Help_Over.jpg');
end;

procedure TFrmMenuUI.ImgHelpMouseLeave(Sender: TObject);
begin
 imgHelp.Picture.LoadFromFile('tdGame\images\buttons\Help.jpg');
end;

//Menu fader
procedure TFrmMenuUI.FadeTimerTimer(Sender: TObject);
begin
 if FrmMenuUI.AlphaBlendValue >= 255
   then FadeTimer.Enabled := false
   else FrmMenuUI.AlphaBlendValue := FrmMenuUI.AlphaBlendValue + 3;
end;
end.
```

5.2. LoadUI:

```
unit LoadUI;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, jpeg, ExtCtrls, DBU, ComCtrls, MMSystem;

type
  TFrmLoadUI = class(TForm)
    ImgLoadGame: TImage;
    ImgLoad: TImage;
    ImgDelete: TImage;
    ImgDeleteAll: TImage;
    LstLoad: TListBox;
    procedure ImgLoadMouseEnter(Sender: TObject);
    procedure ImgLoadMouseLeave(Sender: TObject);
    procedure ImgDeleteMouseEnter(Sender: TObject);
    procedure ImgDeleteMouseLeave(Sender: TObject);
    procedure ImgDeleteAllMouseEnter(Sender: TObject);
    procedure ImgDeleteAllMouseLeave(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure ImgDeleteClick(Sender: TObject);
    procedure LstLoadClick(Sender: TObject);
    procedure ImgDeleteAllClick(Sender: TObject);
    procedure ImgLoadClick(Sender: TObject);
  private
    Procedure Load;
  public
    { Public declarations }
  end;

var
  FrmLoadUI: TFrmLoadUI;
  Index: Integer;
implementation

{$R *.dfm}

uses MenuUI;
```

```
//Menu option mouse effects
procedure TFrmLoadUI.Load;
Var
 LoadSQL, Temp, Time, Name: String;
begin
 LstLoad.Clear;
 LstLoad.Items.Add('Saved Time:                    Saved Name:');

 LoadSQL:= 'SELECT SavedTime, SavedName FROM tblSavedGame';
 Change(LoadSQL);
 MyDb.Open;
 MyDb.First;
 While not myDB.Eof do
  Begin
   Time:= MyDb.Fields.FieldByName('SavedTime').AsString;
   Name:= MyDb.Fields.FieldByName('SavedName').AsString;
   Temp:= Time+'     '+Name;
   LstLoad.Items.Append(Temp);
   MyDB.Next;
  end;
end;

//Sets up the form using the load procedure.
procedure TFrmLoadUI.FormShow(Sender: TObject);
begin
 Load;
end;

Function TimeExtract(FLine: String): String;
Var
 Posi: Integer;
begin
 Posi:= Pos('M',FLine);
 Result:= Copy(FLine,1,Posi)
end;

//Selection based save deletion.
procedure TFrmLoadUI.ImgDeleteClick(Sender: TObject);
Var
 Save, DeleteSQL: String;
begin
 If Index = 0
  Then Showmessage('Invalid Selection')
  Else
   If (MessageDlg('Delete Selected Save?'+' Yes or No?'+#13+'Deleted save will be lost forever!',
    mtConfirmation, [mbYes, mbNo],0) = mrYes)
     Then begin //Deletion based on SavedTime.
      Save:= TimeExtract(LstLoad.Items[Index]);
      DeleteSQL:= 'DELETE * FROM tblSavedGame WHERE SavedTime LIKE "'+Save+'"';
      Change(DeleteSQL);
      Load;
     end;

end;
```

```
//All save deletion.
procedure TFrmLoadUI.ImgDeleteAllClick(Sender: TObject);
Var
 DeleteSQL: String;
begin
 If (MessageDlg('Delete All Saves?'+' Yes or No?'+#13+'Deleted saves will be lost forever!',
   mtConfirmation, [mbYes, mbNo],0) = mrYes)
    Then begin //Deletes every save.
     DeleteSQL:= 'DELETE * FROM tblSavedGame';
     Change(DeleteSQL);
     Load;
    end;
end;

procedure TFrmLoadUI.ImgDeleteAllMouseEnter(Sender: TObject);
begin
 ImgDeleteAll.Picture.LoadFromFile('tdGame\images\buttons\Delete_All_Over.jpg');
end;

procedure TFrmLoadUI.ImgDeleteAllMouseLeave(Sender: TObject);
begin
 ImgDeleteAll.Picture.LoadFromFile('tdGame\images\buttons\Delete_All.jpg');
end;

procedure TFrmLoadUI.ImgDeleteMouseEnter(Sender: TObject);
begin
 ImgDelete.Picture.LoadFromFile('tdGame\images\buttons\Delete_Over.jpg');
end;

procedure TFrmLoadUI.ImgDeleteMouseLeave(Sender: TObject);
begin
 ImgDelete.Picture.LoadFromFile('tdGame\images\buttons\Delete.jpg');
end;

procedure TFrmLoadUI.ImgLoadClick(Sender: TObject);
Var
 Load: String;
 isLoad: Boolean;
begin
 isLoad:= False;
 If Index = 0
   Then Showmessage('Invalid Selection')
   Else
    If (MessageDlg('Load the selected save?'+' Yes or No?'+#13+TimeExtract(LstLoad.Items[Index]),
     mtConfirmation, [mbYes, mbNo],0) = mrYes)
      Then begin //Deletion based on SavedTime.
       Load:= TimeExtract(LstLoad.Items[Index]);
       FrmMenuUI.BuildGame(isLoad,Load);
       sndPlaySound(nil, SND_ASYNC or SND_LOOP);
       Hide;
      end;
end;
```

```
procedure TFrmLoadUI.ImgLoadMouseEnter(Sender: TObject);
begin
  ImgLoad.Picture.LoadFromFile('tdGame\images\buttons\Load_Over.jpg');
end;

procedure TFrmLoadUI.ImgLoadMouseLeave(Sender: TObject);
begin
  ImgLoad.Picture.LoadFromFile('tdGame\images\buttons\Load.jpg');
end;

//Recieve Listbox selection index.
procedure TFrmLoadUI.LstLoadClick(Sender: TObject);
begin
  Index:= LstLoad.ItemIndex;
end;
end.
```

5.3. GameUI:

```
unit GameUI;

interface

uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
 Dialogs, jpeg, ExtCtrls, StdCtrls, HelpUI, QuestUI, BaseU, CharacterU, MapU,
 ComCtrls, MMSystem;

type
 TFrmGameUI = class(TForm)
  ImgStatsBar: TImage;
  LblCharName: TLabel;
  LblHP: TLabel;
  LblAT: TLabel;
  ImgQuest: TImage;
  ImgExit: TImage;
  ImgHelp: TImage;
  LblLvl: TLabel;
  ImgMap: TImage;
  ImgChar: TImage;
  TimOppAI: TTimer;
  TimATControl: TTimer;
  PBATP: TProgressBar;
  TimAnimator: TTimer;
  TimAttack: TTimer;
  procedure ImgQuestMouseEnter(Sender: TObject);
  procedure ImgQuestMouseLeave(Sender: TObject);
  procedure ImgHelpMouseLeave(Sender: TObject);
  procedure ImgHelpMouseEnter(Sender: TObject);
  procedure ImgExitMouseEnter(Sender: TObject);
  procedure ImgExitMouseLeave(Sender: TObject);
  procedure ImgHelpClick(Sender: TObject);
  procedure ImgExitClick(Sender: TObject);
  procedure FormKeyUp(Sender: TObject; var Key: Word; Shift: TShiftState);
  procedure ImgCharMouseEnter(Sender: TObject);
  procedure ImgQuestClick(Sender: TObject);
  procedure TimATControlTimer(Sender: TObject);
  procedure FormShow(Sender: TObject);
  procedure TimOppAITimer(Sender: TObject);
  procedure TimAnimatorTimer(Sender: TObject);
  procedure TimAttackTimer(Sender: TObject);
 private
  { Private declarations }
 public
  Procedure UpdateForm;
  Procedure SortImgArr(PIndex:Integer);
  Procedure StandardMov(PKey: Char);
  Procedure DiagMov(PKey: Char);
 end;
```

```
var
  FrmGameUI: TFrmGameUI;
  ImgArr: Array[1..6] of TImage;
  Cnt: Integer;
  Timer: Integer;
  AITimer: Integer;
  AniCnt, AttCnt: Integer;
  Direction: String;
implementation

{$R *.dfm}

  Uses EntityU;
```

Procedure TFrmGameUI.StandardMov(PKey: Char);

```
begin
     Case PKey of
     'A':Direction:= 'Left';
     'D':Direction:= 'Right';
     'W':Direction:= 'Up';
     'S':Direction:= 'Down';
     End;
     TimAnimator.Enabled:= True;
     UpdateForm;
end;
```

procedure TFrmGameUI.DiagMov(PKey: Char);

```
begin
     Case PKey of
     'Q':Direction:= 'Left';
     'E':Direction:= 'Right';
     'Z':Direction:= 'Left';
     'C':Direction:= 'Right';
     End;
     TimAnimator.Enabled:= True;
     UpdateForm;
end;
```

```
procedure TFrmGameUI.FormKeyUp(Sender: TObject; var Key: Word;
 Shift: TShiftState);
begin
 If Upcase(chr(key)) IN ['A','W','S','D']
 Then If (TheMap.CalcMovment(chr(key)) = true) AND (TimAttack.Enabled = False)
  Then StandardMov(chr(key));

 If Upcase(chr(key)) IN ['Q','E','Z','C']
 Then If (TheMap.CalcDiagMovment(chr(key)) = true) AND (TimAttack.Enabled = False)
   Then DiagMov(chr(key));

 IF Upcase(chr(key)) = 'I'
  Then If TheMap.CalcItem = true
   Then UpdateForm;

 IF Upcase(chr(key)) = 'O'
  Then If (TimATControl.Enabled = False) AND (TheCharacter.GetATPoints > 0)
   Then begin
    TimATControl.Enabled:= True;
    TimAttack.Enabled:= True;
    ImgChar.Picture.LoadFromFile('tdGame\images\chars\Attack\1.bmp');
    ImgChar.Left:= ImgChar.Left - 50;
    ImgChar.Top:=  ImgChar.Top - 50;
    TheEntity.CalcCharAction;
    PBATP.Position:= 0;
   end;
end;

procedure TFrmGameUI.FormShow(Sender: TObject);
begin
  ImgChar.Picture.LoadFromFile('tdGame\images\chars\Walking\Right\1.bmp');
  TimOppAI.Enabled:= True; //Initiates constant A.I. running.
  Timer:= 20;   //Players AT speed.
  AITimer:= 40; //Opps AT Speed.
  PBATP.Position:= timer;
  Cnt:= 0;
  AniCnt:= 4;
  AttCnt:= 5;
  sndPlaySound('tdGame\music\EverWinterKnightsPlay.wav', SND_ASYNC or SND_LOOP);
end;

procedure TFrmGameUI.ImgCharMouseEnter(Sender: TObject);
begin
 //ImgChar.ShowHint:= True;
 //ImgChar.Hint:=('Left: '+inttostr(TheCharacter.GetXPos)+'Top: '+inttostr(TheCharacter.GetYPos))
 //for debuging
end;
```

```
procedure TFrmGameUI.ImgExitClick(Sender: TObject);
begin
 If (MessageDlg('Are you sure you wish to exit?'+' Yes or No?'+#13+'Any unsaved progress will be lost!',
  mtConfirmation, [mbYes, mbNo],0) = mrYes)
  Then begin
   sndPlaySound(nil, SND_ASYNC or SND_LOOP);
   TheCharacter.Destroy;
   TheMap.Destroy;
   TheEntity.Destroy;
   Application.Terminate
  end;
end;

procedure TFrmGameUI.ImgExitMouseEnter(Sender: TObject);
begin
 ImgExit.Picture.LoadFromFile('tdGame\images\buttons\In_Game_Exit_Over.jpg');
end;

procedure TFrmGameUI.ImgExitMouseLeave(Sender: TObject);
begin
 ImgExit.Picture.LoadFromFile('tdGame\images\buttons\In_Game_Exit.jpg');
end;

procedure TFrmGameUI.ImgHelpClick(Sender: TObject);
begin
 FrmHelpUI.Show;
end;

procedure TFrmGameUI.ImgHelpMouseEnter(Sender: TObject);
begin
 ImgHelp.Picture.LoadFromFile('tdGame\images\buttons\In_Game_Help_Over.jpg');
end;

procedure TFrmGameUI.ImgHelpMouseLeave(Sender: TObject);
begin
 ImgHelp.Picture.LoadFromFile('tdGame\images\buttons\In_Game_Help.jpg');
end;

procedure TFrmGameUI.ImgQuestClick(Sender: TObject);
begin
 FrmQuestUI.Show;
end;

procedure TFrmGameUI.ImgQuestMouseEnter(Sender: TObject);
begin
 ImgQuest.Picture.LoadFromFile('tdGame\images\buttons\In_Game_Quest_Over.jpg');
end;

procedure TFrmGameUI.ImgQuestMouseLeave(Sender: TObject);
begin
 ImgQuest.Picture.LoadFromFile('tdGame\images\buttons\In_Game_Quest.jpg');
end;
```

```
procedure TFrmGameUI.SortImgArr(PIndex: Integer); //Matches opp image arr to opp class array.
Var
 Loop: Integer;
 Temp: TImage;
begin
 //Showmessage(inttostr(PIndex)); //Debugging;
 ImgArr[PIndex].Destroy; //Destroy opponent image.
 For loop:= PIndex+1 to Cnt do
   begin
    Temp:= ImgArr[loop];
    ImgArr[loop-1]:= temp;
   end;
    Dec(Cnt);
end;

procedure TFrmGameUI.TimAnimatorTimer(Sender: TObject);
begin
 If (AniCnt <> 4)
   Then begin
    Inc(AniCnt);
   ImgChar.Picture.LoadFromFile('tdGame\images\chars\Walking\'+Direction+'\'+inttostr(Anicnt)+'.bmp');
   end
   Else begin
    AniCnt:= 0;
    TimAnimator.Enabled:= False;
   end;
end;

procedure TFrmGameUI.TimATControlTimer(Sender: TObject);
begin
   If Timer = 0
    then begin
    TimATControl.Enabled:= False;
    timer:= 20;
    UpdateForm;
    end
    else begin
      dec(timer);
      PBATP.Position:= PBATP.Position + 1
    end;
end;
procedure TFrmGameUI.TimAttackTimer(Sender: TObject);
begin
 If AttCnt <> 5
   Then begin
    Inc(AttCnt);
    ImgChar.Picture.LoadFromFile('tdGame\images\chars\Attack\'+inttostr(AttCnt)+'.bmp');
    If Attcnt = 5
      Then begin
       TimAttack.Enabled:= False;
       ImgChar.Picture.LoadFromFile('tdGame\images\chars\Walking\'+Direction+'\1.bmp');
       ImgChar.Top:= ((25*TheCharacter.GetYPos)+10);
       ImgChar.Left:=((25*TheCharacter.GetXPos)-25);
      end;
   end
   Else AttCnt:= 0;
end;
```

```
procedure TFrmGameUI.TimOppAITimer(Sender: TObject);
begin
 If AITimer = 0
  Then begin
   TimOppAI.Enabled:= True; //continues timer.
   AITimer:= 40;
   TheEntity.CalcOppAction;
   UpdateForm;
  end
  else dec(AITimer)
end;

//Updates all changable details on the form.
procedure TFrmGameUI.UpdateForm;
begin
 With FrmGameUI do
  begin
   LblCharName.Caption:= TheCharacter.GetNaam;
   LblHP.Caption:= inttostr(TheCharacter.GetLife);
   LblLvl.Caption:= inttostr(TheCharacter.Getlevel);
   LblAT.Caption:= inttostr(TheCharacter.GetATPoints);
   ImgMap.Picture.LoadFromFile('tdMaps/'+(inttostr(TheCharacter.GetMap))+'.JPG');
   ImgChar.Top:= ((25*TheCharacter.GetYPos)+10);
   ImgChar.Left:=((25*TheCharacter.GetXPos)-25);
  end;
end;
end.
```

5.4. HelpUI:

```
unit HelpUI;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, jpeg, ExtCtrls, StdCtrls, ComCtrls;

type
  TFrmHelpUI = class(TForm)
    ImgHelp: TImage;
    RchGameHelp: TRichEdit;
    procedure FormCreate(Sender: TObject);
  private
  public
    { Public declarations }
  end;

var
  FrmHelpUI: TFrmHelpUI;
implementation

{$R *.dfm}

procedure TFrmHelpUI.FormCreate(Sender: TObject);
Var
  RamFile: TextFile;
  Line: String;
  Size: Integer;
begin
  Size:= 0;
  AssignFile(RamFile,'tdGame\help\Help.txt');
  If FileExists('tdGame\help\Help.txt')
    Then Reset(RamFile)
    Else Rewrite(RamFile);
  Reset(RamFile);
  While not EOF(RamFile) do
    begin
      ReadLN(RamFile,Line);
      RchGameHelp.Lines.Add(Line)
    end;
  CloseFile(RamFile);
end;

end.
```

5.5. QuestUI:

```
unit QuestUI;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ComCtrls, jpeg, ExtCtrls, CharacterU, DBU;

type
  TFrmQuestUI = class(TForm)
    ImgQuest: TImage;
    LstCompQuests: TListBox;
    RchActiveQuest: TRichEdit;
    procedure FormShow(Sender: TObject);
  private
    Procedure GetActiveQuest;
    Procedure GetCompletedQuests;
  public
    { Public declarations }
  end;

var
  FrmQuestUI: TFrmQuestUI;

implementation

{$R *.dfm}
procedure TFrmQuestUI.GetActiveQuest;
Var
  QuestNameSQL, QuestDesSQL, QuestExpSQL, QuestAwdSQL: String;
  QuestName, QuestDes, QuestExp, QuestAwd: String;
  StructuredOutcome: String;
begin
  QuestNameSQL:= 'SELECT QuestName FROM tblQuests WHERE QuestNumber LIKE
'''+inttostr(TheCharacter.GetQuest)+'''';
  QuestName:= (Query(QuestNameSQL));

  QuestDesSQL:= 'SELECT QuestDescription FROM tblQuests WHERE QuestNumber LIKE
'''+inttostr(TheCharacter.GetQuest)+'''';
  QuestDes:= (Query(QuestDesSQL));

  QuestExpSQL:= 'SELECT QuestExpLvl FROM tblQuests WHERE QuestNumber LIKE
'''+inttostr(TheCharacter.GetQuest)+'''';
  QuestExp:= (Query(QuestExpSQL));

  QuestAwdSQL:= 'SELECT QuestAwdLvl FROM tblQuests WHERE QuestNumber LIKE
'''+inttostr(TheCharacter.GetQuest)+'''';
  QuestAwd:= (Query(QuestAwdSQL));

  StructuredOutcome:= QuestName+#13+#13+'Quest Description:'+#13+QuestDes+#13+#13+'Expected
Level:'+#9+QuestExp+#13+#13+'Awarded Level:'+#9+QuestAwd;
  RchActiveQuest.Clear;
  RchActiveQuest.Lines.Add(StructuredOutcome);
end;
```

```
procedure TFrmQuestUI.GetCompletedQuests;
Var
 QuestSQL, Quest, Temp: String;
 Loop: integer;
begin
 LstCompQuests.Clear;
 LstCompQuests.Items.Add('Quest Name:                    Saved:');
 For Loop:= 1 to TheCharacter.GetQuest-1 do
  begin
   QuestSQL:= 'SELECT QuestName FROM tblQuests WHERE QuestNumber LIKE "'+inttostr(Loop)+'"';
   Change(QuestSQL);
 MyDb.Open;
 MyDb.First;
 While not myDB.Eof do
  Begin
   Quest:= MyDb.Fields.FieldByName('QuestName').AsString;
   Temp:= Quest+'     '+'Saved';
   LstCompQuests.Items.Append(Temp);
   MyDB.Next;
  end;
 end;
end;

procedure TFrmQuestUI.FormShow(Sender: TObject);
begin
 GetActiveQuest;
 GetCompletedQuests;
end;
end.
```

5.6. MapU

```
unit MapU;

interface

uses CharacterU, DBU, sysutils, Dialogs, Controls;

 Type TMap = class

  Private      //Grid is wider than height. (X;Y) Co-Ordinates
   ObjArr: Array[1..32,1..20] of string; //Holds values of 'Y',''N','I' for each grid position.
   SizeX, SizeY: Integer;  //Numerical values that stores the “size” of the array on the X&Y axis.
   MapName: String;  //Used to identify which map player is located at
  Protected

  Public
   Constructor Create(MNa:String);
   //Accessor
   Function GetMapName: String;
   //Mutator
   Procedure SetMapName(MNa:String);
   //Other
   Procedure Extract(FLine: String);
   Procedure CalcVirtualGrid;
   Function CalcMovment(Direction: Char): Boolean;
   Function CalcDiagMovment(Direction: Char): Boolean;
   Function CalcCollision(FX,FY: Integer): Boolean;
   Function CalcDiagCollision(FX,FY: Integer): Boolean;
   Procedure CalcMapMove(New:String);
   Function CalcIsQuestItem(FRef: String): Boolean;
   Function CalcItem: Boolean;
   Procedure Heal;
 end;

Var
 TheMap: TMap;
implementation

{ TMap }

Uses
 EntityU, GameUI;
```

```
//Extracts each value from a text file to be inserted into the 2D array.
//Used in conjunction with CalcVirtualGrid. WORKS
Procedure TMap.Extract(FLine: String);
var
 Posi: Integer;
begin
 SizeX:= 0;
 While Length(FLine) > 0 do
  begin
   Posi:= pos(',',FLine);
   Inc(SizeX);
   ObjArr[SizeX,SizeY]:= Copy(FLine,1,Posi-1);
   Delete(Fline,1,posi)
  end;
end;

//Builds up the 2D array from a corresponding text file
//WORKS
procedure TMap.CalcVirtualGrid;
Var
 RamFile: TextFile;
 Line: String;
begin
 SizeY:= 0;
 AssignFile(RamFile,'tdMaps/'+(inttostr(TheCharacter.GetMap)+'.txt'));
 If FileExists('tdMaps/'+(inttostr(TheCharacter.GetMap))+'.txt')
  Then Reset(RamFile)
  Else Rewrite(RamFile);
 Reset(RamFile);
 While not EOF(RamFile) do
  Begin
   ReadLN(RamFile,Line);
   Inc(SizeY);
   Extract(Line);
  End;
 CloseFile(RamFile)
end;
```

```
//Used to calculate the characters movement in conjunction with the CalcCollision
function TMap.CalcMovment(Direction: Char): Boolean;
Var
 Xpos,YPos, NewMapNum: Integer;
 MapNameSQL, MapName: String;
begin
 XPos:= TheCharacter.GetXPos;
 YPos:= TheCharacter.GetYPos;
 If Direction = 'W'
  Then Ypos:= Ypos-1
  Else If Direction = 'S'
   Then Ypos:= Ypos+1
   Else If Direction = 'D'
    Then Xpos:= Xpos+1
    Else Xpos:= Xpos-1;
 If CalcCollision(XPos,YPos) = True //Calls collision checker.
  Then Begin
   TheCharacter.SetXPos(Xpos);
   TheCharacter.SetYPos(Ypos);
   If Uppercase(ObjArr[XPos,YPos][1]) = 'M'
    Then begin
     NewMapNum:= strtoint(Copy((ObjArr[XPos,YPos][2]),1,1));;
     MapNameSQL:= 'SELECT MapName FROM tblMaps WHERE MapNumber LIKE
"'+inttostr(NewMapNum)+'"';
     MapName:= (Query(MapNameSQL));
      If (MessageDlg('Travel to: '+MapName+'?',
        mtConfirmation, [mbYes, mbNo],0) = mrYes)
       Then CalcMapMove(ObjArr[XPos,YPos]);
    end;
   Result:= True; //On screen movement can take place.
  End
  Else Result:= False;
end;
```

```
//Used to calculate the characters diagonal movement in conjunction with the CalcDiagCollision
function TMap.CalcDiagMovment(Direction: Char): Boolean;
Var
 Xpos,YPos: Integer;
begin
 XPos:= TheCharacter.GetXPos;
 YPos:= TheCharacter.GetYPos;
 If Direction = 'Q'
  Then Begin
   Ypos:= Ypos-1;
   Xpos:= Xpos-1
  End
  Else If Direction = 'E'
   Then Begin
    Ypos:= Ypos-1;
    Xpos:= Xpos+1
   End
   Else If Direction = 'Z'
    Then Begin
     Ypos:= Ypos+1;
     Xpos:= Xpos-1
    End
    Else begin
      Ypos:= Ypos+1;
      Xpos:= Xpos+1;
    end;
 If CalcDiagCollision(XPos,YPos) = True //Calls collision checker.
  Then Begin
   TheCharacter.SetXPos(Xpos);
   TheCharacter.SetYPos(Ypos);
   Result:= True; //On screen movement can take place.
  End
  Else Result:= False;
end;


//DO NOT ALTER IN ANY WAY (true - false) not (false - true)
//Used to calculate whether a collision will occur for next movement.
//Returns a true or false value.
function TMap.CalcCollision(FX, FY: Integer): Boolean;
begin
 If (Fx >= 1) AND (Fx <= (SizeX))
  Then If (FY >= 1) AND (FY <= (SizeY))
   Then If Uppercase(ObjArr[Fx,Fy]) <> 'N'
   Then If (TheEntity.CalcOppCollision(Fx,Fy) = False)
    Then Result:= True
    Else Result:= False;
end;
```

```
//Used to calculate whether a collision will occur for next DIAGONAL movement.
function TMap.CalcDiagCollision(FX, FY: Integer): Boolean;
begin
 If (Fx >= 1) AND (Fx <= (SizeX))
  Then If (FY >= 1) AND (FY <= (SizeY))
   Then If Uppercase(ObjArr[Fx,Fy]) = 'D'
    Then If (TheEntity.CalcOppCollision(Fx,Fy) = False)
    Then Result:= True
    Else Result:= False;
end;

//DO NOT ALTER IN ANY WAY!!
//Used to change the map the player is travelling to as well as calling the required refreshing of arrays.
procedure TMap.CalcMapMove(New: String);
Var
 NewMapNumber: Integer;
 CopyMapLeng, CopyXleng, CopyYleng: Integer;
 NewXPos, NewYPos: Integer;
 Loop: Integer;
begin
 CopyMapLeng:= pos('(',New);
 NewMapNumber:= strtoint(Copy(New,2,CopyMapLeng-2));

 Delete(New,1,CopyMapLeng-1);

 CopyXleng:= pos(':',New);
 NewXPos:= strtoint(Copy(New,2,CopyXleng-2));

 CopyYleng:= Length(New)-CopyXleng;
 NewYPos:= strtoint(Copy(New,CopyXleng+1,CopyYleng-1));

 //Showmessage(inttostr(NewMapNumber));
 //Showmessage(inttostr(NewXPos));      //Used for debuging!
 //Showmessage(inttostr(NewYPos));

 TheCharacter.SetMap(NewMapNumber);
 TheCharacter.SetXPos(NewXPos);
 TheCharacter.SetYPos(NewYPos);
 CalcVirtualGrid;
 Heal;
 With FrmGameUI do
  begin
   For Loop:= 1 to cnt do
    begin
     ImgArr[loop].Destroy; //Clear array of opp images.
    end;
   Cnt:= 0;
   TimOppAI.Enabled:= False; //While destroying Entity.
  end;

  TheEntity.CalcOpponentArray; //Should call entity create and new opp for new map.
  FrmGameUI.TimOppAI.Enabled:= True; //Re-enables running of opponent procedures.

end;
```

```
procedure TMap.Heal;
Var
 SavedHPSQL: string;
 SavedHP: string;
begin
    SavedHPSQL:= 'SELECT SavedLife FROM tblSavedGame WHERE SavedTime LIKE
"'+TheCharacter.GetSaveID+'"';
    SavedHP:= Query(SavedHPSQL);
    //Showmessage(TheCharacter.GetSaveID);
    //Showmessage('"'+SavedHP+'"');
    If SavedHP <> ''
      Then TheCharacter.SetLife(strtoint(SavedHP));
end;

function TMap.CalcIsQuestItem(FRef: String): Boolean;
Var
 QuestItemSQL, QuestNumberSQL, QuestItem: String;
 QuestNumber: Integer;
begin
 QuestItemSQL:= 'SELECT QuestItemRef FROM tblQuests WHERE QuestNumber LIKE
"'+inttostr(TheCharacter.GetQuest)+'"';
 QuestItem:= (Query(QuestItemSQL)); //What Item does your current quest require?

 QuestNumberSQL:= 'SELECT QuestNumber FROM tblQuests WHERE QuestItemRef LIKE "'+FRef+'"';
 QuestNumber:= strtoint(Query(QuestNumberSQL));

 If (QuestItem = FRef) AND (QuestNumber = TheCharacter.GetQuest) //Is the required quest item the
item in 'passed' position?
   Then Result:= True //Yes it is
   Else Result:= False; //No its not.

end;
```

```
function TMap.CalcItem: Boolean;
Var
 Xpos,YPos: Integer;
 Cell, ItemNameSQL, ItemName, ItemRef, ItemSQL, MessageSQL, TheMessage, QuestSQL, QuestMessage:
String;
 NewATPoints, CheckATP, NewQuest, NewLevel, NewHP: Integer;

begin
 XPos:= TheCharacter.GetXPos;
 YPos:= TheCharacter.GetYPos;
 Cell:= Uppercase(ObjArr[XPos,YPos]);

 If upcase(Cell[1]) = 'I' //Does this position hold an item?
  Then Begin
   ItemRef:= Copy(Cell,2,(Length(Cell)-1)); //What item is held in this position.
   If (CalcIsQuestItem(ItemRef) = True) //Is this Item a quest item?
    Then Begin //Was a quest item.
     QuestSQL:= 'SELECT QuestName FROM tblQuests WHERE QuestNumber LIKE
"'+inttostr(TheCharacter.GetQuest)+'"';
     QuestMessage:= Query(QuestSQL);  //The quest name you have completed.

     ItemNameSQL:= 'SELECT ItemName FROM tblItems WHERE ItemRef LIKE "'+ItemRef+'"';
     ItemName:= Query(ItemNameSQL); //what item have you acquired?

     Showmessage('MISSION COMPLETED!'+#13+QuestMessage+#13+'Item Name: '+ItemName);

     ItemSQL:= 'SELECT ItemATPoints FROM tblItems WHERE ItemRef LIKE "'+ItemRef+'"';
     CheckATP:= strtoint(Query(ItemSQL)); //Does this quest item have an ATP value?

     If (CheckATP <> 0) //Checks that you can't accept an Item with 0 ATP value.
      Then begin
       MessageSQL:= 'SELECT ItemName,ItemATPoints FROM tblItems WHERE ItemRef LIKE
"'+ItemRef+'"';
       TheMessage:= 'Item Name:'+#9+'Item AT Points:'+#13+StructuredQuery(MessageSQL);
       //Details of the item!
       If (MessageDlg('Pick up item?'+#13+TheMessage,
       mtConfirmation, [mbYes, mbNo],0) = mrYes) //Do you want this item?
       Then begin
        NewATPoints:= strtoint((Query(ItemSQL)));
        TheCharacter.SetATPoints(NewATPoints);
        Result:= True;
       end;
      end;

     NewQuest:= (TheCharacter.GetQuest)+1;
     TheCharacter.SetQuest(NewQuest); //Ensures a quest item with o ATP will still activate the next quest.

     NewLevel:= (TheCharacter.GetLevel)+1;
     TheCharacter.SetLevel(NewLevel); //Ensures player always gains level for completing quest.

     NewHP:= (TheCharacter.GetLevel)*(TheCharacter.GetLevel);
     TheCharacter.SetLife(NewHP); //HP is the product of LVL*LVL //Req. Debugging.

     TheEntity.CalcGameWin;
     TheEntity.SaveGame; //Save process after each quest.
    End
    Else Begin //Was not a quest item
     MessageSQL:= 'SELECT ItemName,ItemATPoints FROM tblItems WHERE ItemRef LIKE '''+ItemRef+'''';
     //Is a message and requires structure (#9 & #13)
     TheMessage:= 'Item Name:'+#9+'Item AT Points:'+#13+StructuredQuery(MessageSQL);

     If (MessageDlg('Pick up item?'+#13+TheMessage,
         mtConfirmation, [mbYes, mbNo],0) = mrYes)
       Then begin
         ItemSQL:= 'SELECT ItemATPoints FROM tblItems WHERE ItemRef LIKE '''+ItemRef+'''';
         NewATPoints:= strtoint((Query(ItemSQL)));
         TheCharacter.SetATPoints(NewATPoints);
         Result:= True;
       End;
    end;
end;
end;

//Instantiates the object with the:
//Map Number & Map Name data passed as parameters
constructor TMap.Create(MNa: String);
begin
  MapName:= MNa;
end;

//Returns the Field value
function TMap.GetMapName: String;
begin
  Result:= MapName;
end;

//Changes the Filed value
procedure TMap.SetMapName(MNa: String);
begin
  MapName:= MNa;
end;

end.
```

5.7. EnitiyU:

```
unit EntityU;
//This class holds all the opponent classes.
interface

  uses Sysutils, Dialogs, MapU, CharacterU, OpponentU, BaseU, GameUI, DBU, MMSystem;

  Type TEntity = class

    Private
      OppArr: Array[1..20] of TBase;
      Size: Integer;
      Index: Integer; //Which opponent in array to attack.
    Protected
    Public
      Constructor Create;
      //Other
      Procedure CalcOpponentArray;
      Function CalcOppCollision(FX, FY: Integer): Boolean;
      Procedure CalcCharAction;
      Procedure CalcOppAction;
      Function CalcAttack: Boolean;
      Procedure CalcOppDeath;
      Procedure CalcGameWin;
      Procedure CalcGameLoss;
      Procedure SaveGame;
  end;

Var
  TheEntity: TEntity;

implementation

{ TEntity }

Uses MenuUI;

constructor TEntity.Create;
begin

end;
```

```
procedure TEntity.CalcOpponentArray;
Var
 OppCntSQL, OppMapSQL, OppLVLSQL,OppHPSQL, OppATPSQL, OppXPosSQL, OppYPosSQL :String;
 OppCnt, Loop, OppMap, OppLVL, OppHP, OppATP, OppXPos, OppYPos: Integer;
begin
 loop:= 1;
 Size:= 0;
 index:= 0;
 OppCntSQL:= 'SELECT Count(OppNumber) AS OppCnt FROM tblOpponents';
 OppCnt:= strtoint((Query(OppCntSQL))); //How many opponents are there in total in the table.
 For loop:= 1 to OppCnt do
  Begin
  OppMapSQL:= 'SELECT OppMap FROM tblOpponents WHERE OppNumber LIKE "'+inttostr(loop)+'"';
  OppMap:= strtoint((Query(OppMapSQL)));
   If OppMap = TheCharacter.GetMap //If an opponent in the table has same map then it’s loaded.
    Then begin
     OppLVLSQL:= 'SELECT OppLVL FROM tblOpponents WHERE OppNumber LIKE "'+inttostr(loop)+'"';
     OppHPSQL:= 'SELECT OppHP FROM tblOpponents WHERE OppNumber LIKE "'+inttostr(loop)+'"';
     OppATPSQL:= 'SELECT OppATP FROM tblOpponents WHERE OppNumber LIKE "'+inttostr(loop)+'"';
     OppXPosSQL:= 'SELECT OppXPos FROM tblOpponents WHERE OppNumber LIKE "'+inttostr(loop)+'"';
     OppYPosSQL:= 'SELECT OppYPos FROM tblOpponents WHERE OppNumber LIKE "'+inttostr(loop)+'"';
     OppLVL:= strtoint((Query(OppLVLSQL)));
     OppHP:= strtoint((Query(OppHPSQL)));
     OppATP:= strtoint((Query(OppATPSQL)));
     OppXPos:= strtoint((Query(OppXPosSQL)));
     OppYPos:= strtoint((Query(OppYPosSQL)));
     Inc(Size); //increase size of array.
     OppArr[Size]:= TOpponent.Create(OppLVL, OppHP, OppATP, OppMap, OppXPos, OppYPos);
    //This instantiates each opponent into a specific position in the array.
   end;
  End;
end;

function TEntity.CalcOppCollision(FX, FY: Integer): Boolean;
Var
 Flag:Boolean;
begin
 Index:=1;
 Flag:= False;
  While (Flag = False) AND (Index <= Size) do
   Begin If ((OppArr[Index].GetXPos) = FX) AND ((OppArr[Index].GetYPos) = FY)
    Then Flag:= True
    Else Inc(Index)
   End;

 If Flag = true
  Then Result:= True //Yes you have collided with an opponent
  Else Result:= False; //No you have collided with an opponent
end;
```

```
Function TEntity.CalcAttack: Boolean; //The characters attack.
Var
 Flag: boolean;
begin
 Flag:= False;
 If (TheCharacter.GetXPos-1 <> 0) OR (TheCharacter.GetXPos+1 <> 32)
  Then If (TheCharacter.GetYPos-1 <> 0) OR (TheCharacter.GetXPos+1 <> 21)
   Then Begin
    If CalcOppCollision(TheCharacter.GetXPos-1,TheCharacter.GetYPos) = true
     Then Flag:= True
     Else If CalcOppCollision(TheCharacter.GetXPos+1,TheCharacter.GetYPos) = true
      Then Flag:= True
      Else If CalcOppCollision(TheCharacter.GetXPos,TheCharacter.GetYPos-1) = true
       Then Flag:= True
       Else If CalcOppCollision(TheCharacter.GetXPos,TheCharacter.GetYPos+1) = true
        Then Flag:= True;
   End;
 Result:= Flag;
end;

procedure TEntity.CalcOppAction;
begin
 If CalcAttack = True
  Then Begin
   TheCharacter.SetLife(TheCharacter.GetLife - OppArr[Index].GetATPoints);
   CalcGameLoss;
  End;
end;

procedure TEntity.CalcCharAction;
begin
 If CalcAttack = True
  Then begin
   OppArr[Index].SetLife(OppArr[Index].GetLife - TheCharacter.GetATPoints);
   If OppArr[Index].GetLife < 1
    Then CalcOppDeath;
  end;
end;

procedure TEntity.CalcOppDeath;
Var
 loop: Integer;
 Temp: TBase;

begin
 OppArr[Index].Destroy; //Destroy opponent class.
 For loop:= Index+1 to size do
  begin
   Temp:= OppArr[loop];
   OppArr[loop-1]:= temp;
  end;
   Dec(size);
   FrmGameUI.SortImgArr(Index);
end;
```

```
procedure TEntity.CalcGameLoss;
Var
 Loop: Integer;
begin
 If TheCharacter.GetLife < 1
  Then Begin
   FrmGameUI.TimOppAI.Enabled:= False; //Must happen first.
   FrmGameUI.TimAttack.Enabled:= False;
   FrmGameUI.TimAnimator.Enabled:= False;
   FrmGameUI.ImgChar.Picture.LoadFromFile('tdGame/images/chars/CharDeath.bmp');
   FrmGameUI.ImgChar.Left:= FrmGameUI.ImgChar.Left - 25;
   FrmGameUI.ImgChar.Top:= FrmGameUI.ImgChar.top + 25;
   Showmessage('Active Quest Failed');
   TheCharacter.Destroy;
   TheMap.Destroy;
   TheEntity.Destroy;
   With FrmGameUI do
   begin
    For Loop:= 1 to cnt do
     begin
      ImgArr[loop].Destroy; //Clear array of opp images.
     end;
    Cnt:= 0;
  end;
   FrmGameUI.Hide; //Or maybe just reload from previous save.
   sndPlaySound(nil, SND_ASYNC or SND_LOOP);
   FrmMenuUI.Show;
  End;
end;


//Only called when completing a quest.
procedure TEntity.CalcGameWin;
Var
 CheckSQL: String;
 GameWin: Integer;
begin
 CheckSQL:= 'SELECT Count(*) AS FinalQuest FROM tblQuests';
 GameWin:= strtoint(Query(CheckSQL));
 //Showmessage(inttostr(GameWin));
 //Checks to see if the player's quest is the same as that of the final quest
 If TheCharacter.GetQuest = GameWin+1
  Then begin
   Showmessage('Congratulations on finishing Ever Winter Knights'+#13+'Finally after decades of winter
you have restored summer');
   TheMap.CalcMapMove('I5(1:18)');
   FrmGameUI.UpdateForm;
  end;

end;
```

```
procedure TEntity.SaveGame;
Var
 CheckSQL, SaveSQL, TempID: String;
 SavedTime: String; //Used for a new save!
begin
 SavedTime:= DateToStr(Date)+' '+TimeToStr(Time);

 CheckSQL:= 'SELECT SavedTime FROM tblSavedGame WHERE SavedTime Like
"'+TheCharacter.GetSaveID+'"';
 TempID:= (Query(CheckSQL));

 //Checks to see if player has previous save
 If TempID = TheCharacter.GetSaveID
 Then begin //Will only update their previous save
  SaveSQL:= 'UPDATE tblSavedGame SET ' +
  'SavedTime = "'+SavedTime+'" ,' +
  'SavedName = "'+TheCharacter.GetNaam+'" ,' +
  'SavedQuest = "'+inttostr(TheCharacter.GetQuest)+'" ,' +
  'SavedLocation = "'+inttostr(TheCharacter.GetMap)+'" ,' +
  'SavedLevel = "'+inttostr(TheCharacter.GetLevel)+'" ,' +
  'SavedLife = "'+inttostr(TheCharacter.GetLife)+'" ,' +
  'SavedATPoints = "'+inttostr(TheCharacter.GetATPoints)+'" ,' +
  'SavedXPos = "'+inttostr(TheCharacter.GetXpos)+'" ,' +
  'SavedYPos = "'+inttostr(TheCharacter.GetYpos)+'" WHERE SavedTime LIKE
"'+TheCharacter.GetSaveID+'"';
  Change(SaveSQL)
 end
 Else
  begin
   SaveSQL:= 'INSERT INTO tblSavedGame
VALUES("'+SavedTime+'","'+TheCharacter.GetNaam+'","'+inttostr(TheCharacter.GetQuest)+'","'+inttostr(TheCharacter
.GetMap)+'","'+inttostr(TheCharacter.GetLevel)+'","'+inttostr(TheCharacter.GetLife)+'","'+inttostr(TheCharacter.GetAT
Points)+'","'+inttostr(TheCharacter.GetXPos)+'","'+inttostr(TheCharacter.GetYPos)+'")';
   Change(SaveSQL)
  end;
  TheCharacter.SetSaveID(SavedTime);
  Showmessage('Game Saved!');
end;
end.
```

5.8. BaseU:

```
unit BaseU;
//This class is the basic structure of both the character and the opponent classes.
interface

Type TBase= class

    Private
    Protected
      Level, Life, ATPoints, Map, XPos, YPos: Integer ; //Inherited fields
    Public
      Constructor Create(Lvl,HP,ATP,M,XP,YP: Integer);
      //Accessor
      Function GetLevel: Integer;
      Function GetLife: Integer;
      Function GetATPoints: Integer;
      Function GetMap: Integer;
      Function GetXPos: Integer;
      Function GetYPos: Integer;
      //Mutator
      Procedure SetLevel(Lvl: Integer);
      Procedure SetLife(HP: Integer);
      Procedure SetATPoints(ATP: Integer);
      Procedure SetMap(M: Integer);
      Procedure SetXPos(XP: Integer);
      Procedure SetYPos(YP: Integer);
  end;

Var
  TheBase: TBase;

implementation

{ TBase }

constructor TBase.Create(Lvl, HP, ATP, M, XP, YP: Integer);
begin
  Level:= lvl;
  Life:= HP;
  ATPoints:= ATP;
  Map:= M;
  Xpos:= XP;
  YPos:= YP;
end;

//Accessor Methods
function TBase.GetATPoints: Integer;
begin
  Result:= ATPoints;
end;

function TBase.GetLevel: Integer;
begin
  Result:= Level;
end;
```

```
function TBase.GetLife: Integer;
begin
  Result:= Life;
end;

function TBase.GetMap: Integer;
begin
  Result:= Map;
end;

function TBase.GetXPos: Integer;
begin
  Result:= XPos;
end;

function TBase.GetYPos: Integer;
begin
  Result:= YPos;
end;

//Mutator Methods
procedure TBase.SetATPoints(ATP: Integer);
begin
  ATPoints:= ATP;
end;

procedure TBase.SetLevel(Lvl: Integer);
begin
  Level:= Lvl;
end;

procedure TBase.SetLife(HP: Integer);
begin
  Life:= HP;
end;

procedure TBase.SetMap(M: Integer);
begin
  Map:= M;
end;

procedure TBase.SetXPos(XP: Integer);
begin
  XPos:= XP;
end;

procedure TBase.SetYPos(YP: Integer);
begin
  YPos:= YP;
end;

end.
```

5.9. CharacterU:

```
unit CharacterU;
//The character class which inherits fields from the Base Class.
interface

 uses BaseU, Sysutils, Dialogs;

 Type TCharacter = class(TBase)
  Private
   SaveID: String; //A date & time ID value used to check if player has past save.
   Naam: String;
   Quest: Integer;
  Protected
  Public
   Constructor Create(Lvl,HP,ATP,M,XP,YP,Q: Integer; Na,ID: String);
   //Accessor
   Function GetSaveID: String;
   Function GetNaam: String;
   Function GetQuest: Integer;
   //Mutator
   Procedure SetSaveID(ID: String);
   Procedure SetNaam(Na: String);
   Procedure SetQuest(Q: Integer);
 end;

Var
 TheCharacter: TCharacter;

implementation

{ TCharacter }

constructor TCharacter.Create(Lvl,HP,ATP,M,XP,YP,Q: Integer; Na,ID: String);
begin
 Inherited Create(Lvl,HP,ATP,M,XP,YP);
 SaveID:= ID;
 Naam:= Na;
 Quest:= Q;
end;

//Accessor Methods
function TCharacter.GetNaam: String;
begin
 Result:= Naam;
end;

function TCharacter.GetQuest: Integer;
begin
 Result:= Quest
end;

function TCharacter.GetSaveID: String;
begin
 Result:= SaveID;
end;
```

```
//Mutator Methods
procedure TCharacter.SetNaam(Na: String);
begin
  Naam:= Na;
end;

procedure TCharacter.SetQuest(Q: Integer);
begin
  Quest:= Q;
end;

procedure TCharacter.SetSaveID(ID: String);
begin
  SaveID:= ID;
end;

end.
```

5.10. OpponentU:

```
unit OpponentU;
//The opponent class which inherits fields from the Base Class.
interface

  uses BaseU, Sysutils, Graphics, jpeg,  GameUI, ExtCtrls;

  Type TOpponent = class(TBase)
    Private
    Protected
      XPos, YPos: Integer;
      Enemy: TBitmap;
    Public
    Constructor Create(Lvl,HP,ATP,M,XP,YP: Integer);

  end;

Var
  AnOpponent: TOpponent;

implementation

{ TOpponent }

constructor TOpponent.Create(Lvl, HP, ATP, M, XP, YP: Integer);
begin
  Inherited Create(Lvl,HP,ATP,M,XP,YP);
  Xpos:= ((25*XP)-25);
  YPos:= ((25*YP)+15);
  With FrmGameUI do
    begin
      Inc(Cnt);
      ImgArr[cnt]:= TImage.create(FrmGameUI);
      ImgArr[Cnt].Parent:= FrmGameUI;
      ImgArr[Cnt].Picture.LoadFromFile('tdGame/images/opponents/'+inttostr(Lvl)+'.bmp');
      ImgArr[cnt].AutoSize:= True;
      ImgArr[cnt].Transparent:= True;
      ImgArr[cnt].Left:= Xpos;
      ImgArr[cnt].Top:= YPos;

    end;
end;

end.
```

5.11. DBU:

```
unit DBU;

interface

uses DB, ADODB, forms;

var MyDB : TADOQuery;

procedure opendb;
Function Query(TSQL: String): String;
Function StructuredQuery(TSQL: String): String;
Procedure change(TSQL: String);

implementation

procedure opendb;
begin
  MyDB := TADOQuery.Create(Application);
  MyDB.ConnectionString := 'Provider=Microsoft.Jet.OLEDB.4.0;Data Source=' +
    '"tdGame/GameDB.mdb";Persist Security Info=False'; //Used to link to MS access database
end;


Procedure change(TSQL: String);
begin
  begin
    MyDB.Close;
    MyDB.SQL.Text:= TSQL;
    MyDB.ExecSQL;
  end;
end;

Function StructuredQuery(TSQL: String): String;
Var
  temp: String;
  Loop: Integer;
begin
  temp:='';
    Change(TSQL);  //Essential before a new SQL statement is used. Closes the Data-Set
    MyDb.Open;  //Open Data-Set so you can use it.
    MyDb.First; //Internal pointer set to first data-set.
    While not MyDB.Eof do //EOF -End of File.
    Begin //Extracting each field, by heading.
      For loop:= 1 to MyDB.FieldCount do
        Temp:= Temp+MyDB.Fields.FieldByNumber(loop).AsString+#9;
        MyDb.Next;
        Temp:= Temp+#13
    End;
  Result:= temp;
end;
```

```
Function Query(TSQL: String): String;
Var
 temp: String;
 Loop: Integer;
begin
 temp:='';
  Change(TSQL);  //Essential before a new SQL statement is used. Closes the Data-Set
  MyDb.Open;  //Open Data-Set so you can use it.
  MyDb.First; //Internal pointer set to first data-set.
  While not MyDB.Eof do //EOF -End of File.
  Begin //Extracting each field, by heading.
   For loop:= 1 to MyDB.FieldCount do
    Temp:= Temp+MyDB.Fields.FieldByNumber(loop).AsString;
    MyDb.Next;
  End;
 Result:= temp;
end;

end.
```

6. Bibliography / Referencing:

Images:

- All images used in the project were created by me using Photoshop CS4.
- Creatures and animation schemes developed by http://spritemoviemaker.deviantart.com/gallery/#Sprites

Music:

- Final Fantasy 13 Official Soundtrack.

Internet Sources:

- Paton, J. © 2010. Understanding Delphi, http://www.jesararts.net/delphi.php , March 2010 *(Personal Site) – Previous practical's used.*
- Spirito De, E. © 2000. Playing a custom sound, http://www.latiumsoftware.com/en/delphi/00024.php , March 2010.

People who have contributed:

- Mrs. Emmett – Help with classes, inheritance and storing of data.
- Stephanie Courtnage – Helped in creating concept for storing map object information in text file.
- Mathew Yoko – For help with coded generate of TImage fields for enemies.
- Clint Rogers – General troubleshooting and concept help.