BREAKOUT

Alden Luc R. Hade
De La Salle University-Manila
2401 Taft Avenue,
Manila, Philippines 1004
alden_luc_hade@dlsu.edu.ph

1. INTRODUCTION

A text-based game (not to be confused with texting games) is a video game that relies on text characters in order to be functional, as opposed to vector or bitmap graphics[1]. There is some ambiguity whether text graphics are needed (e.g. ASCII art) or if a text-based user interface is enough. A text-based user interface revolves around the use of text strings to issue commands to the game.

Escape the room games are a subgenre of text-based and point-and-click games which require a player to utilize their environment to escape from imprisonment[2]. The popularity of these games has led to the development of real-life escape rooms around the world, such as Breakout Philippines.

In the spirit of these escape rooms, BREAKOUT is a text-based game developed as part of the requirements for the CSC617M course in De La Salle University-Manila. BREAKOUT is a text-based game in which the goal is to escape imprisonment by finding clues and items in the environment.

The source code for BREAKOUT is available at https://github.com/aldsTM/BREAKOUT.

2. DESIGN AND MECHANICS

2.1. Gameplay

The game contains visual elements from point-and-click adventure games, such as the cutaway view of the room and the objects within it. However, control of the game is through text commands rather than mouse clicks. The player is stuck in a room, and at each step of the way, hints are given regarding the current status of the game. The player then must enter a command consisting of a verb and optionally an object in order to control the flow of the game. An example of a command is

> get flashlight

The game is helpful and does not assume player proficiency in text-based games. Incomplete commands will result in the game asking questions in order to get the player to complete the command correctly, such as

> pick up
What do you pick up?
> flashlight
What do you do with the flashlight?
> pick up flashlight
You retrieve the flashlight.

Such is the nature of earlier text-based games: they usually have little room for ambiguity, and this is by design, since most of the earlier text-based games are role-playing games which have complex mechanics; therefore, incomplete commands may result in the computer executing player attacks with the wrong type of weapon, for example.

2.2. Control Scheme

At every turn, the player is tasked with typing commands. The mouse is needed to click on the text field, while the keyboard is used to type in the commands.

2.3. List of Commands

The following list is sorted alphabetically.

1. **Connect.** Synonyms include *attach* and *link*.
2. **Dial.** Alternatively, the player can type in *call*.
3. **Examine.** Synonyms include *search, examine, loot,* or simply *check*.
4. **Look for.** The player can also type in *search for, watch, find,* or *discover*.
5. **Pick up.** Also interpreted as *collect, get, acquire, retrieve,* and *obtain*.
6. **Plug.** This is parsed the same way as *plug in*.
7. **Read.** The player can also type in *see* or *look at*.
8. **Shout.** Try using it! Either type in this or *help*.
9. **Use.** Different from examine. Allows the player to make use of inert objects. Some alternatives are *turn on, utilize,* or simply *on*.
10. **Wait.** Used for some situations where there is no need to do an action. This is interpreted in the same vein as *be patient, patience, continue, ok, okay,* or *proceed*.
11. **Walk.** Used for some situations wherein the character does not necessarily move automatically. Synonyms include *start walking, move, inch forward,* or *go*.

2.4. List of Objects

In the English language structure, objects are entities that are acted on by the subject; or, simply, the objects are the things to which, or with which, verbs are done[3]. For example, in the sentence,

Tom studies grammar.

studies is the verb and the object is *grammar*.

Not all of the listed commands in Section 2.3 require an object, because not all of them are transitive verbs. (Transitive verbs require an object; intransitives do not.) The following is a list of objects with which the player can interact in the game.

1. **Cellular phone.** Typed in as either *phone, cellphone, cell,* or *smartphone*.
2. **Charger.**
3. **Corpse.** Typed in as either *corpse, body, dead man, or dead person.*
4. **Flashlight.** Can be typed as-is or using the British English equivalent *torch*.
5. **Letter.** Also recognized by the game as a *note*.
6. **Location.** Refers to *location data*, or *GPS coordinates*. Can also be typed in as *location info*.
7. **Police.** Can also be called *cops, 911, 117,* or *emergency*.
8. **Socket.** The player can also type in *outlet*.

3. IMPLEMENTATION

BREAKOUT was implemented using Microsoft PowerPoint and Visual Basic for Applications. Mainly used as a presentation program, PowerPoint was created in 1987 by a company called Forethought, Inc. PowerPoint was later bought by Microsoft. The Office 95 release of PowerPoint introduced the ability for users to utilize VBA for advanced functionality.

VBA, or Visual Basic for Applications, is an implementation of the event-driven programming language Visual Basic 6, which was supported by Microsoft[4]. It can be used to control many parts of the application to which it is linked (in this case PowerPoint), such as manipulating user interface elements and allowing users to interact on-the-fly with forms or dialog boxes.

Microsoft PowerPoint was chosen purely for ease-of-use. Code for a linear text-based game can be repetitive, and the style of Visual Basic accommodates ease when coding with this limitation in mind. Each slide represents a slice of time when the user needs to input a command.

3.1. Code Structure

Code in VBA is divided into two types, at least for this implementation: [1] Object-based code and [2] module-based code. Object-based code is code that is written for a specific slide. In this program, there are 21 slides, of which 19 are interactive. Therefore, there are 19 files of code embedded in the PowerPoint presentation which each control a specific slide. The following are the subroutines in each "file":

1. **ansTextBox_KeyDown().** This is the main code that happens per slide. It checks the user input and compares it to the lists of global variables declared in the module-based code. Each slide represents a slice of time, so for each slide, only a couple of commands at maximum is considered correct. For example, for the first slide, the only correct commands are

   walk forward *(or any variation thereof)*
   or
   search room *(or any variation thereof)*

2. **ansTextBox_GotFocus().** Purely aesthetic, this only controls the clearing of the textbox when the user clicks on it.

Module-based code, meanwhile, is code that can run irrespective of slide. Module-based code was used in this program solely to store global variables that should be preserved across slides, which are the names of the commands and objects, respectively. Module-based code can be thought of as the "dictionary" of this text-based game, wherein all the synonyms of all the objects and commands are stored for checking in every slide.

3.2. Input Checking

Input was checked using the following algorithm:

```
commandIsCorrect = false
When (enter)
  String input = scan user input;
  Split(input) into parts;
  if parts(0) matches correctVerbKeyword {
    if (parts(lastElement or secondToLastElement)
matches correctObjectKeyword {
      commandIsCorrect = true
    }
  }
```

This assumes that the first word is a verb, and the second-to-last OR last word is an object. The "matching" is done by checking if the keyword is one of the synonyms of the correct verb (each correct verb is an array; so the algorithm loops through the synonyms of this keyword).

Not all input challenges require the user to put a verb-object pair (e.g. "walk" can be written as-is). For these cases, only the correctVerbKeyword is checked.

This code assumes that different users can type in differently-varying numbers of words in their command. The parser only cares about the verb and the object.

4. SCREENSHOTS AND SNIPPETS

4.1. Title Screens

The game starts and ends with screens that are designed to evoke the feel of bigger games. However, these are mostly set dressing.

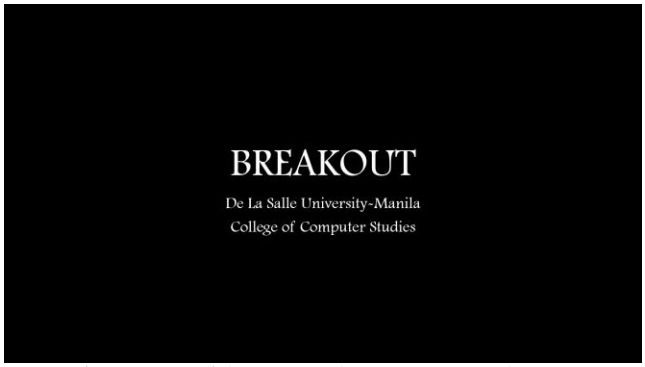

Figure 4.1. Title Screen that appears on bootup.

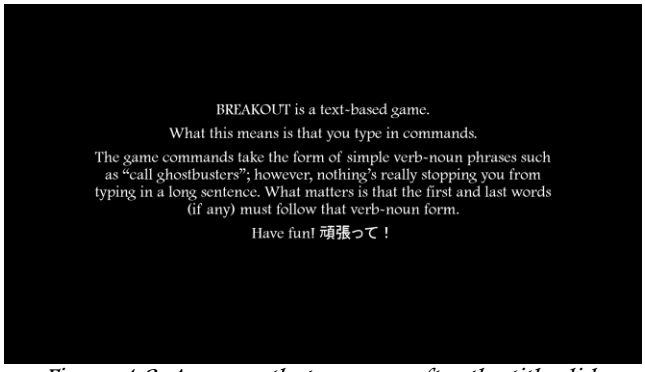

Figure 4.2. A screen that appears after the title slide designed to quickly introduce players to the game.

4.2. The Game

The game takes place in a dark room. In every step of the way, hints will appear at the bottom of the screen narrating the current situation as well as providing hints on what to do. The text area is supposed to be clicked before it can be typed on.

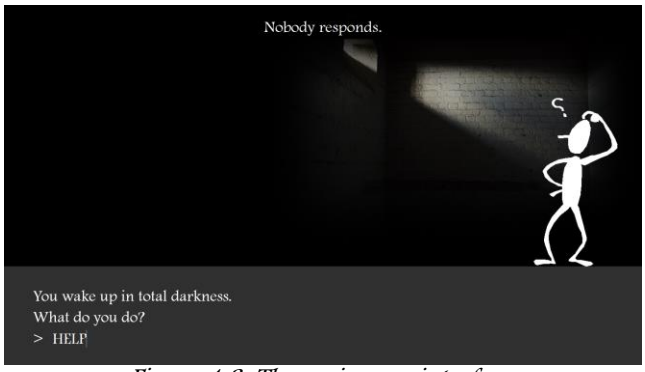

Figure 4.3. The main user interface.

Assuming the player writes a wrong input, the game will throw immersive "error messages" on top of the screen which represent the character's state of mind. These provide hints as to what the player should be typing in order to proceed. As mentioned earlier in Section 2.1, the game assumes no player expertise with text-based games, which is why it is designed to be helpful.

5. IN CONCLUSION

While the activity was rewarding and fun, the development definitely encountered some issues regarding design and implementation.

The primary problem and realization of this activity is the nature of the inputs. The inputs are written by humans and so can be worded in any number of ways which are all equivalent in human understanding. For example, one might say, "I don't care," while another might say, "I couldn't care less," and they both mean the same thing. For computers, obviously, such strings are very different. Therefore, for this exercise, the implementation was done with the limitation that the only parts that the code cares about in the input are the [1] verb and [2] the object (if applicable). So long as the required parts are present, the user can type in as long or as convoluted a string as they like. Filler words are ignored so long as the input is correct.

The other main challenge and realization with this project is time constraints in a creative project. Designing a game is usually a time-consuming process. Coupled with implementation, this means that designing a good game that is programmatically sound requires lots of time and effort.

All in all, BREAKOUT is a fun game that gives insight into human grammar as well as gives a throwback to old text-based RPGs and escape-the-room games.

6. REFERENCES

1. Sweetster (2008). *Emergence in Games.*
2. divisionten (2009). *Escape-the-Room Games: A History, Catalogue, and Explanation – Kino Diaries.* Accessed at http://www.theotaku.com/worlds/kinodiaries/view/122787/escape-the-room_games%3A_a_history,_a_catalogue,_and_an_explanation/.
3. Conner, J. (1968). *A grammar of standard English.* Boston: Houghton Mifflin Company.
4. *Microsoft Developer Network.* Accessed at *msdn.microsoft.com.*
5. Hade, A. (2018). *BREAKOUT.* Accessed at https://github.com/aldsTM/BREAKOUT.