Blind Warcraft

by Elliott Ploutz, Angelo Dejesus, Brandon Navarro, Trenton Balogh, Jehoshua Josue

Introduction

Our intention is to design a suite of tools that can be installed onto the World of Warcraft video game allowing visually impaired and blind users to play the game independently. The team is using the game's front facing API and add-on support to adjust the interface to better accommodate players who cannot use the traditional interface system to receive information. We plan to create and modify existing programs to allow for an expansive text to speech program that presents as much information as possible through audio cues, an improved interface system, and scripts which allow for independent movement and combat without outside help.

Executive Summary

Video game developers rarely design games to be accessible to visually impaired or blind users and when they do, they tend to focus specifically on that audience. The issue is there aren't very many games that people with and without eyesight can play together. Due to the large community of people playing World of Warcraft, there also exists a community of players who have enjoyed the game prior to losing their eyesight, or simply wish to be able to play the game with their friends in general, but are unable to do so due to the game's interface and controls. Our team is interested in exploring how to translate this 3D environment into a game that is playable without eyesight while still maintaining player choice and overall fun.

We are creating a series of additions that can be installed directly into the game as well as expanding work on an already begun open source project that runs alongside World of Warcraft to allow for blind accessible play. Quest text can be read through a screen reader that runs outside of the main game to allow players to understand their objectives. Other add-ons will be created to notify the user when a quest is nearby and how to receive it.

Almost all of the game's default interfaces only utilize mouse input which is difficult for a blind user to use without assistance. Several of these interfaces will be recreated with the add-on in order to allow for keyboard control navigation. The information displayed in these interfaces will also be voiced for the user to allow them to be able to understand and navigate the individual interfaces effectively. The inventory will be redesigned to better accommodate blind users by providing audio information regarding the bag and highlighting any upgrades the player can make regarding their equipment. These inventory commands can be controlled through command prompt input in game. The spellbook will work in a similar manner to the inventory system, by allowing users to select which spells they would like to bind to keyboard keys through keyboard commands and text to speech assistance. Combat will also require information regarding player and enemy health/mana to be output through audio.

Through expanding the audio output of the game to provide more information regarding the 3D space, we hope to provide a playable experience that a blind user could utilize without any outside direction within the playable portion of the game. Also on expanding the command prompts usability, we hope to allow users to be able to have full control over our software within the game.

Requirements

The Functional Requirements

Since our add-ons will be passive adjustments to the game, our interface will be limited to what the game will allow us. The standard way of accessing the functionality will be to use “slash commands” through the in-game chat window. For example, the player can type “/bw bag info” and the application will activate the bag functionality and run the bwBagInfo() function, returning the name of the first item in their inventory. Once a particular area has been activated, items, questing, spells, etc. the player can navigate by manually entering /bw next and /bw prev or using the up and down arrow keys on the keyboard. Note, these are the default layout, but are also customizable by the player.

The functional table is broken into four parts by the ID of F# where the # is:

I - item
Q - questing
S - Spellbook

ID	Title	Requirement Description	Date	Done
F1	Up Arrow Key	In the below features (bag, quest, spellbook), this key will toggle to the previous object of interest.	10/6/15	**X**
F2	Down Arrow Key	In the below features (bag, quest, spellbook), this key will toggle to the next object of interest.	10/6/15	**X**
FI1	/bw bag	The bag command is the start of the item feature. It reads the title of the first item. When entered a second time, it deactivates the functionality.	10/6/15	**X**
FI2	/bw bag next	Gets the next item in the bag.	10/6/15	**X**
FI3	/bw bag prev	Gets the previous item in the bag.	10/6/15	**X**
FI4	/bw bag more	Reads the current item’s information, including whether it’s ready or unavailable, the “cool down,” and description.	10/6/15	**X**
FI5	/bw bag item [number]	Gets the item at the given number in the bags.	10/6/15	**X**
FI6	/bw bag bind [key]	Binds the current item to a key on the toolbar.	10/30/15	**X**

FI7	/bw bag destroy	Deletes the current item.	10/30/15	**X**
FI8	/bw bag equip	Equips the current item if possible, then resets the bag functionality.	11/15/15	**X**
FS1	/bw spellbook /bw sb	The spellbook command is the start of the abilities feature. It reads the title of the first ability in the spellbook. If this command is entered again, it will deactivate the feature.	10/20/15	**X**
FS2	/bw spellbook next /bw sb next	There are multiple pages or panes in the spellbook, this allows the user to navigate to the next spell.	10/20/15	**X**
FS3	/bw spellbook prev /bw sb prev	This allows the user to navigate to the previous spell.	10/20/15	**X**
FS4	/bw spellbook more /bw sb more	Returns the current spell's information, including cast time, the "cool down," range, and description.	10/20/15	**X**
FS5	/ bw spellbook bind [key] /bw sb bind [key]	Binds the current spell to a hotkey on the keyboard.	10/20/15	**X**
FQ1	/bw quest count	Prints the number of quests in your quest log	10/9/15	**X**
FQ2	/bw quest [number]	Assigns active quest. Prints the title and completion status of the quest.	10/10/15	**X**
FQ3	/bw quest stored or [keybinding]	Prints your currently active quest.	10/10/15	**X**
FQ4	/bw quest abandon	Abandon currently active quest.	10/17/15	**X**
FQ5	/bw quest progress or [keybinding]	Prints progress of currently active quest	10/17/15	**X**
FQ6	/bw quest select	Allows user to scroll through their quest log, printing the title and completion status of the quest. The last viewed quest will be selected.	10/17/15	**X**

When bag, spellbook, or quest features are called, there will be a queue like data structure that reads through the information.

F1-F2: The bag, quests, and spellbooks text navigation will require navigation keys when sorting through the list of spells or equipment. This is needed in order to determine whether or not you want to investigate the next item on the list or perform some sort of action on the currently selected item. Depending on the type of filter you placed, it will skip any unasked for query. In short, this will be the keyboard equivalent of highlighting over the item with a traditional mouse.

FI1: Starts the item management add-on and will utilize the text to speech software to read off bag items.

FI2: Sets the next item as the current and reads the name.

FI3: Provides general filtering of objects. If used on trash items, the text to speech will only read non-trash items when listing off the contents of your bag.

FI4: Provides all information beyond the item name from the currently selected item through text to speech software.

FI5: Sets the currently selected item to the index passed.

FI6: Provides the same functionality as dragging an item from your bag onto the toolbar.

FI7: Calls the API to delete the current item in the inventory. It bypasses the confirmation box which asks whether the item should really be destroyed.

FI8: Equips the current item to the character and resets the bag functionality. The reset ensures that the equipped item doesn't cause the other functions to crash on the empty spot. If the item can't be equipped, the game has the character voice that it cannot equip it.

FS1: Opens the spellbook and begins listing off the first spell through text to speech.

FS2: The next command goes to the numerical next spell in the spellbook.

FS3: The previous command goes to the numerical previous spell in the spellbook.

FS4: Returns the current spell's information, including cast time, the "cool down," range, and description in text to speech format.

FS5: The player can input a key on the keyboard to bind the spell to that key. When the specified key is pressed, that spell will activate if applicable.

FQ1: Returns the amount of active quests in your quest log. This is also used for error checking other questing functions internally.

FQ2: Used to directly assign the active quest if the user knows the index of the quest, otherwise they will use /bw quest select. The selected quest will be saved to a global variable that is always saved, even between play sessions, in the .toc file.

FQ3: Because many of the other quest functions automatically operate on the currently selected quest, it is important for the user to know what their selected quest is. Users can type this command in to see the title of their selected quest. This function can also be accessed through keybinding for convenient access.

FQ4: Automatically abandons the quest whose index is currently saved as the currently selected quest.

FQ5: Returns the progress of your current quest. Example: 0/6 boars killed. This information could also be accessed through a keybound button to be accessed quicker. The information also automatically appears whenever a quest objective is updated.

FQ6: Sets the up and down arrow keys onto quest select mode. When activated, the up and down arrow keys can be used to cycle through the quests in your quest log. Hitting the down arrow key displays the title of the next quest on your log as well as its completion status, the up key will return you to the previous quest. Continuously hitting the down arrow key will eventually cycle you back to the first quest, the opposite applies for the up arrow key. Hitting the numpad plus key will select that quest.

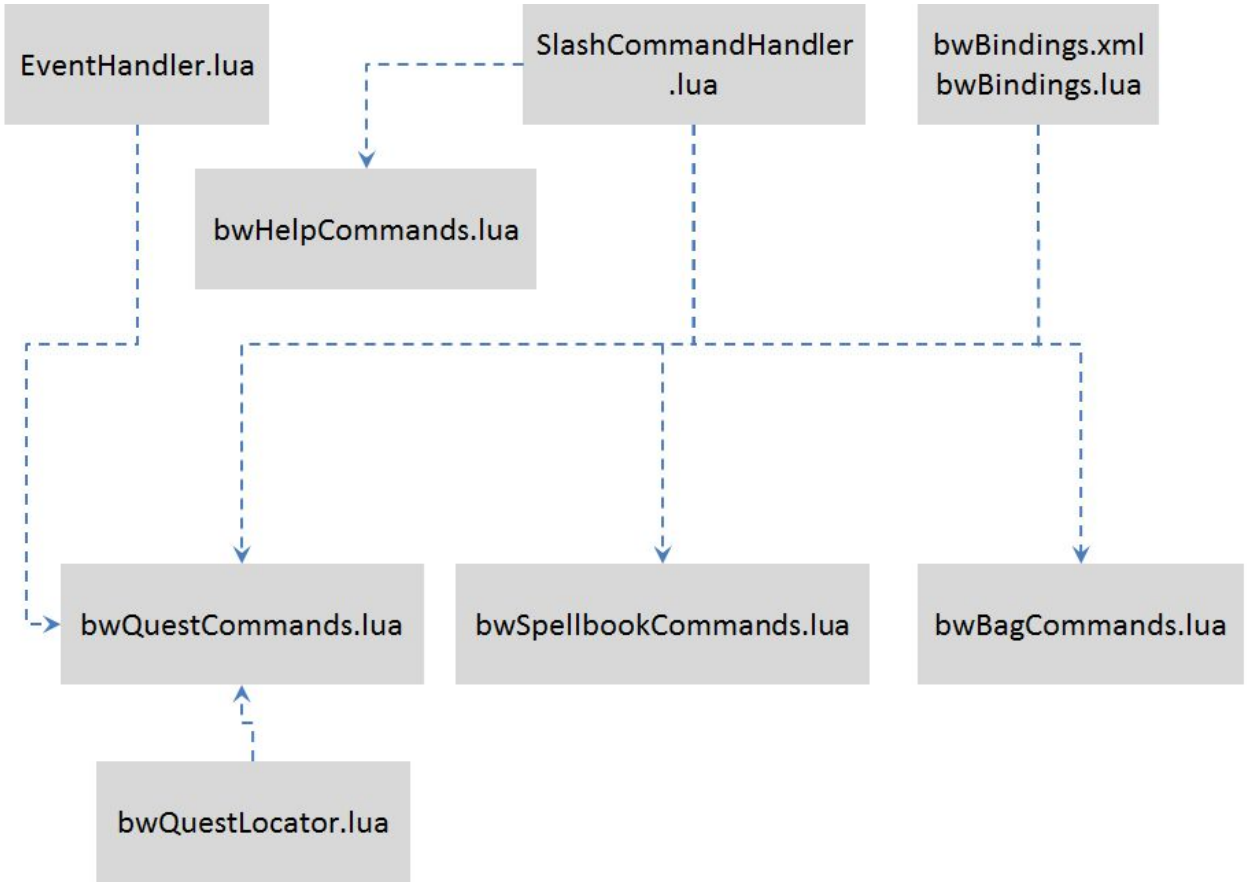
Non-Functional Requirements

ID	Title	Requirement Description
NF1	Operating System	The application will be designed and tested on Windows 7, 8.1, and 10. Other operating systems are not supported at this time.
NF2	Programming Language	LUA and XML are the most common languages used in World of Warcraft add-ons and will be used for this application.
NF3	Open Source AddOns	Open-Source add-ons available on Curse.com that we have reviewed for design techniques are sellTrash, autoEquipQuestItem, and autoRepair.
NF4	JocysCom	An open source text-to-speech application for World of Warcraft.

Design

Most of this project is written in lua, which is a scripting language. As such, the language does not have or support classes. This means that lua does not lend itself well to the standard UML diagram design. Below is an overview of our project, which shows an abstract design. This diagram is expanded upon in the next few pages where instead of having each class as a UML diagram, we have decided to make each individual file into a UML diagram.

UML Diagram



SlashCommandHandler.lua
+bwSlashCommandHandler(msg: string, editbox: string) +bwLoadCommandList() +bwError(errorString : string)

The Slash Command Handler is the object that reads the slash commands entered via the in-game chat box and catches the arguments given. The bwSlashCommandHandler function itself is the most important part as it contains all of the argument cases that our AddOn currently handles (e.g., /bw spellbook, /bw select quest). The bwLoadCommandList function is an internal function that simply associates our command codes "/blindwarcraft" and "/bw" with the list of possible commands. The error function is called when there are errors within the handler function and will give various error messages.

EventHandler.lua
PLAYER_REGEN_DISABLED() PLAYER_REGEN_ENABLED() UNIT_HEALTH(unitID) UNIT_COMBAT(unitID, action, descriptor, damage, damageType) UNIT_TARGET(unitID) UNIT_QUEST_LOG_CHANGED(unitID) QUEST_ACCEPTED(questIndex, questID) QUEST_COMPLETE()

The Event Handler UML is not a set of functions but instead a set of events that our event handler will listen for and respond accordingly. The Event Handler is configured to notify the player when they enter combat which is triggered by the PLAYER_REGEN_DISABLED event. This is because, while in combat, passive player health regeneration is disabled. The PLAYER_REGEN_ENABLED is fired upon the player's regeneration being enabled again, which would happen upon leaving combat. At the moment, the ENABLED event only resets flags. While in combat, the utility is configured to notify the player when they take damage that puts them below 75%, 50%, and 25% health. The percentage health value is calculated when UNIT_HEALTH is fired, which is fired whenever a unit's health changes. The UNIT_COMBAT event is fired when some unit takes damage or healing. In this case, the code checks to make sure that the unit is either the player or the player's target, and notifies the player accordingly. At this time, the player is notified of a hit on the target when it is a Critical hit, Crushing blow, or a Heal. The player is also notified when they are healed and are told the percentage of their health after the heal. The UNIT_TARGET event is fired whenever some unit's target changes. The unitID of this event is the unit that is affected, meaning that if the player changes targets,

the unitID reads “player”. This can be used to detect when the player changes targets as well as getting information about that target like name, level, etc. UNIT_QUEST_LOG_CHANGED is used to detect if the player’s active quest has changed and will update their progress and other variables accordingly. The QUEST_ACCEPTED event is used to detect when the player has accepted a quest in order to add it to the array of quests currently stored. The QUEST_COMPLETE event is fired when the player completes a quest and is used primarily in selecting a reward from the quest giver. While there are other events that we listen for, the ones listed above are much more important to core gameplay than things like a level up notification.

bwBindings.lua and Bindings.xml

```
BW_NAVIGATE_NEXT
BW_NAVIGATE_PREV
BW_NAVIGATE_UP
BW_NAVIGATE_DOWN
BW_STOPREADING
BW_MOREINFO
BW_QUESTDIR
BW_SELECTEDQUEST
BW_SELECTEDQUEST_PROGRESS
```

Bindings.xml and bwBindings.lua work in tandem to allow us to create custom key bindings with custom functions for the game. The lua file is used mainly to define the names of the key bindings so that they appear in the game properly, while the xml file contains all of the actual code that is to be performed. In this case the BW_NAVIGATE bindings allow us to specify the navigation keys for our various interfaces like the spellbook and inventory. BW_MOREINFO is used by the interfaces to fetch more information about the particular item, spell, or quest in question. The BW_STOPREADING binding is there for the player to halt any reading that the text to speech program is currently doing. BW_QUESTDIR is used to direct the player to the selected quest. BW_SELECTEDQUEST is used to learn the name of the currently active quest in case the player forgets and BW_SELECTEDQUEST_PROGRESS is there so the player can tell how far along in the quest’s objectives they are.

bwBagCommands.lua
BWBAGMODE: boolean BAG: integer BAGINDEX: integer ITEMSLOTS: integer table ITEMCOUNT: integer NUMOFITEMS: integer ITEMNAME: string
+bwBagPrev(): void +bwBagNext(): void +bwBagInfo(): const void +bwBagMoreInfo(): const void +bwBagItemAtIndex(int ind): void +bwBagBindItem(): const void +bwBagUseItem(): void -bwBagActivate(): void

The bag functionality is initialized by *bwBagActivate()* which sets all the global variables to their initial values.

BWBAGMODE := Set to true to make the bag functionality active.

BAG := The bag at which the item is in.

BAGINDEX := The first element in ITEMSLOTS.

ITEMSLOTS := The indices found by iterating over the bags by API functions.

ITEMCOUNT := 1

NUMOFITEMS := The size of ITEMSLOTS.

ITEMNAME := The name of the current item.

The rest of the functions are now available for use. The functions *bwBagPrev()* and *bwBagNext()* update the ITEMCOUNT and BAGINDEX values to point to the previous or next item to be viewed. The function *bwBagInfo()* uses the voice-to-text functionality to speak the name of the current item. The function *bwBagMoreInfo()* speaks the additional information normally provided when hovering over the item. The function *bwBagItemAtIndex(int ind)* updates the ITEMCOUNT and BAGINDEX to the passed index if it is within range and valid. The function *bwBagBindItem()* binds the current item to a key on the toolbar. The function *bwBagUseItem()* uses the current item.

bwSpellbookCommands.lua
CURRSBINDEX: integer SBOFFSET: integer MAXSBINDEX: integer BWSBMODE: bool SPELLNAME: string SPELLDESC: string SPELLCASTTIME: string SPELLMINRANGE: string SPELLMAXRANGE: string SPELLID: string
+bwSBBind(integer : key): void +bwSBGiveInfo(bool : full): void +bwSBMoreInfo(): void -bwSBGetInfo(): void -bwSBNavigateNext(): void -bwSBNavigatePrev(): void -bwSBActivate(): void

The spellbook utility is very similar to the inventory utility and will be used manipulate the player's spellbooks and action bars. *bwSBActivate()* is called to initialize the MAXSBINDEX, SBOFFSET, and CURRSBINDEX variables and set BWSBMODE to true. It is called when the user types in "/bw sb", "/blindwarcraft spellbook", or any combination thereof. *bwSBNavigateNext()* and *bwSBNavigatePrev()* are, as their name implies, used to navigate between the spells in the spellbook, incrementing or decrementing the index variables as necessary. While the navigation functions do have commands to call them in the form of "/bw sb next", they are intended to be called via a keypress in order to make navigating the spellbook less clunky. *bwSBGetInfo()* and *bwSBGiveInfo()* are two functions that work in tandem to relay information to the player. The Get function obtains all of the information about the current spell and stores it in all of the SPELL variables, while the Give function will take that information and relay it to the player in a format specified by a boolean. The Give function's boolean is usually false, which would only cause the Give function to display the name of the spell. The only way for the Give function boolean to be true is if the Give function is called via *bwSBMoreInfo().* If the Give function is called in this manner, all of the relevant spell information will be displayed, including its description, cast time, min and max range, and the level that spell is learned. The *bwSBBind()* function allows the player to bind the current spell in their spellbook to a specified key through the command "/bw sb bind #" where the number is the slot on their action bar to which they wish to bind the spell.

bwQuestCommands.lua
SELECTEDQUEST: integer BWQUESTMODE: bool QINDEX: integer QUESTCOUNT: integer = nil
+bwReturnQuestCount(): integer +bwSelectQuest(integer : questSelection): void +bwReadStored(): void +bwAbandonQuest(): void +bwQuestProgress(): void -bwCycleActivate(): void -bwQuestNext(): void -bwQuestPrev(): void

bwReturnQuestCount() calls the WoW API function to return the amount of quests in the player's quest log and stores it into the global variable QUESTCOUNT for debugging and error checking purposes on other functions. The function could also be called on by the player to see the number as well through slash commands as specified in functional requirement FQ1. *bwSelectQuest()* is called by *bwQuestNext()* and *bwQuestPrev()* to select the currently active quest. Players can also call this function if they happen to know the index of the quest that they wish to select without having to cycle to it. When called, either by other functions or in the game (req. FQ6), it will return the title of the quest and whether or not it has been completed as well as store the index of the quest into SELECTEDQUEST. *bwReadStored()* can be bound to any key and will be used to tell the player the title of their active quest while they play. The active quest will be taken from the index stored in SELECTEDQUEST and the player can retrieve this information through the a bound key or slash command "/bw quest stored" *bwAbandonQuest()* will abandon the currently active quest stored in SELECTEDQUEST. *bwQuestProgress()* returns the player's quest progress, this function is also set up so that it could be bound to any key as well as returns automatic updates to the player as they progress through the quest. *bwCycleActivate()* turns on the quest selection mode by setting BWQUESTMODE to true and sets QINDEX to the currently selected quest. QINDEX will either increment or decrement as the user hits the up or down arrow keys which call *bwQuestPrev()* and *bwQuestNext()*, respectively. When QINDEX changes, the index will be passed to the *bwSelectQuest()* function and the title of the quest will be sent to the user and SELECTEDQUEST will be updated to QINDEX's value. Because of this, the user will only need to stop scrolling in order to confirm their quest selection. To set BWQUESTMODE to false, the user must either retype the "/bw quest select" command or enter into another navigation mode such as inventory or spellbook mode.

bwQuestLocator.lua
TARGETEXISTS: integer DISPX1: float DISPX2: float DISPY1: float DISPY2: float DISP_PRESSCOUNTER: integer qDist1: float qDist2: float qDist3: float x1: float x2: float y1: float y2: float questBearing: float pressCount: float xPos: float yPos: float rightOrLeft: integer baseCaseX: float baseCaseY: float facing: float distance: float numerator: float denominator: float sideA: float
+calcQuestBearing(): void -distFormula(float : x1, float : y1, float : x2, float : y2): float -resetVariables(): void -hasQuest(funcParam : bool): bool

The current method is to obtain the coordinates and orientation of the character unit along with the coordinates of the target unit or area. Based on a triangle formed by the initial distance from the target point, the character's displacement, and the final distance from the target point, a bearing is calculated. This bearing reveals the general direction of the target point. Once the bearing is calculated, the user will be prompted to keep turning right until the user is facing the general direction of the target point.

bwHelpCommands.lua

`command: string`
`args1: string`
`args2: string`

`+bwHelp(string : command, string : args1, string : args2): void`

The bwHelp function will take in 3 possible strings. The command string will always be “help”. While the other 2 strings may contain other strings based on the other functions in our command handler. By default, if “/bw help” is used, the function will return several subjects to choose from. As of this time, it is either “bindings”, “spellbook”, “quest” or “read”. If the previous quoted words were used in args1, the function will then display several arguments that may be used in args2. For example “/bw help spellbook” will return to the user possible strings that may be used in args2 such as “next”, ”prev”, or ”more”. If the bwHelp function takes in all 3 working arguments, the function will respond to the user with a short description of what the command does. Meaning, “/bw help sb more” will return an explanation of what “sb more” does.

Prototype

Example User Interfaces

Figure 1 - Here is a an example of the command "/bw quest select" enter into the chatbox. Our LUA script sends a whisper to our character, which is detected by the Jocys text-to-speech program. Jocys in turn reads the message received out loud.

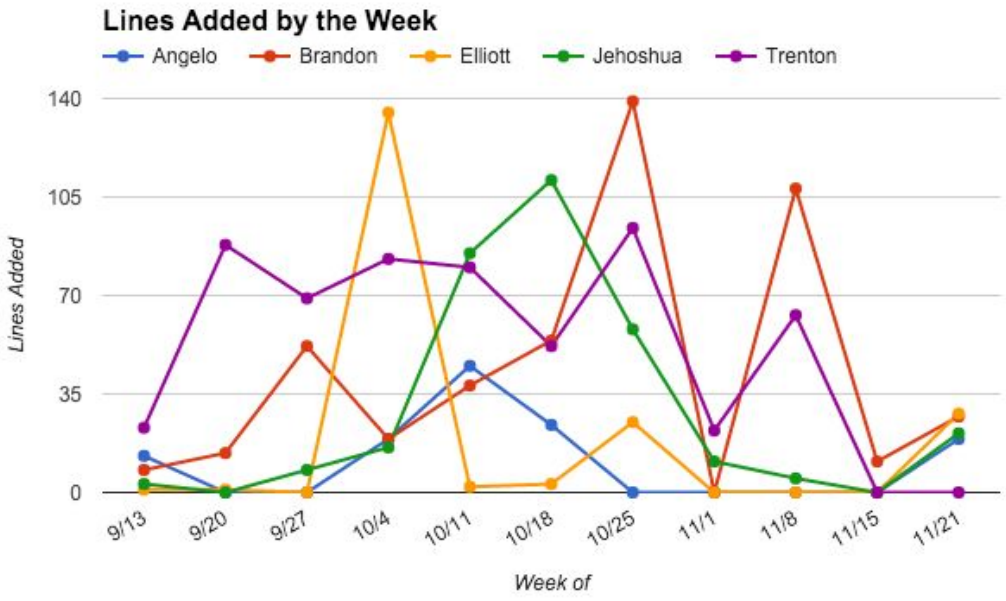
Figure 2 - The interaction between blind user and our LUA script occurs within the chatbox of the World of Warcraft. It was not necessary to create a GUI.

Figure 3 - The two lines emerging from the left side represent the distances of the character from the quest. Angle **α** is calculated by our algorithm. We tell the user to turn around until their bearing is within +/- 0.3 radians of the angle **α**

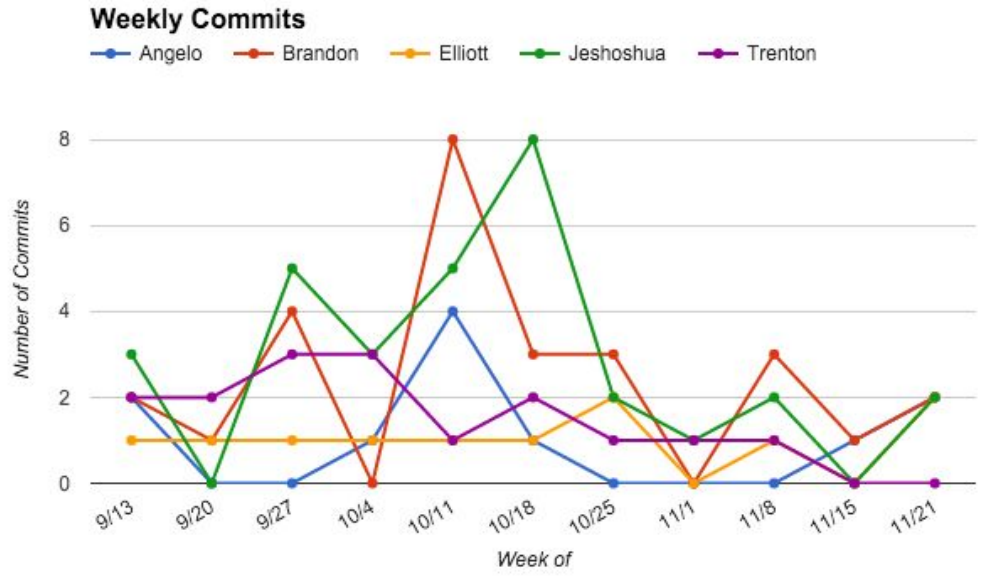
Conclusion

By using the various open source programs and game-add ons, we created a way for visually impaired and blind users to play World of Warcraft with limited to no help when inside of the game. This will be done by creating tools with the intention of making the game more accessible by removing the need to navigate the various interfaces with a mouse pointer as well as providing more audio based information rather than the standard graphics already use.

Graphs



The Lines Edited graph is a representation of the total lines added since the beginning of the start of our Git Assignment program. The weeks were defined by starting on Sunday and ending on Saturday. The only lines not taken into account was from the Jocys's add-on, as that contribution was outside of the group.



Contributions

Trenton Balogh:

Trenton has found various resources for Lua programming, including tutorials as well as references for the World of Warcraft api. He also wrote a couple examples of Lua scripts for the team to use as examples of Lua code. This is in addition to being the one behind most of the general information about World of Warcraft itself. He wrote the UML design document and reviewed and edited the main design paper.

Once the project really got started, Trenton developed the spellbook utility and designed some code related to action bar manipulation. This is in addition to working on the design and implementation of the combat utility.

After the spellbook was finished, Trenton and Elliott began working on a character panel, but sadly, it could not be finished before the end of the semester.

Jehoshua Josue:

Jehoshua helped in finding a Hello World tutorial for Lua and an example of a World of Warcraft add-on. He created the mockup images for the prototype section of the Design Document. He was the first to create the “Hello World” test add-on written in Lua. He also came up with the algorithm for the navigation part of the project. He created the UML diagram that contained the components and dependencies of the project, and improved some aesthetics of the Design Document.

In the weeks that ensued, he worked on creating the algorithm for calculating the correct bearing of the current quest. He was responsible for researching the Lua functions that Blizzard allowed to be used in relation with the user’s position in the game’s virtual world. The algorithm he developed involved the Law of Cosines and triangles. Jehoshua made the function that would calculate an adjustment bearing and then tell the user to turn in order to face the general direction of a quest or a quest giver. He also created the function that would tell the user if the character moved or not by calculating its displacement.

Angelo Dejesus:

Initially combined the separate parts of each user’s initial contributions of the portfolio to one combined document. The editing & formatting of the portfolio, as well as the meeting minutes. He assisted in the presentation slides by adding screenshots, and the graphs that provide information about the commits and codes of line added.

With the ongoing weeks, Angelo has implemented a help function for the users. The help function will link the other developer's’ projects into a more cohesive unit with available descriptions to further the user’s knowledge of the add-on. Also, every time the user logs in, they will be reminded to use /bw help command in case they forget. Additionally he has also

created a functions that allows the user to keybind keys via the command line. As time moves forward he will provide further descriptions to the utilities added into Blind Warcraft. With the end of the coming semester, he has laid the groundwork for users that create a new character and log into the game, it will automatically set the bindings for Blind Warcraft.

Brandon Navarro:

Brandon found an open source text to speech project for World of Warcraft that we will become involved in that allows for chat messages to be read out loud through a separate text to speech client. He also helped design the main features of the project as well as planning the implementations. Brandon is also responsible for the various summaries and explanatory paragraphs in the project portfolio.

As main development began, he designed and began implementing the questing features of the game. This includes getting the information presented in the main interface and sending it to the text to speech, such as the quest titles and progress. He also converted much of the questing interaction that requires the mouse into text and keyboard commands to make them more accessible. The functional requirements and the Quest Utility portion of the UML were updated by him to include his design of these features.

Elliott Ploutz:

Elliott keeps track of the meeting minutes, hosts and controls the Google shared folder, and has found two textbooks on World of Warcraft AddOn programming for the team. All team members have helped to design the utilities, but Elliott expanded on the chat function interface. Elliott has also aided in the feature and design document and in editing the portfolio as a whole. Currently, Elliott is in communication with the owner of www.blindwarcraft.com, a website for blind World of Warcraft players, to gain application testers and insight. Elliott has written the UML and description of the inventory functionality and also implemented it in the application.

Meeting Minutes

9/2/15

Members Attended

Trenton Balogh, Jehoshua Josue, Angelo Dejesus, Brandon Navarro, Elliott Ploutz

Progress Reports

First meeting. We discussed the issues in developing an interface for the blind and partially blind. The issues were broken down into 4 categories, movement, attacking, items, and quests.

Goals for Next Meeting (*For Week 2*):

Trenton: Familiarize with the Lua language and assess add-ons.

Angelo: Think of particular add-ons that may help the visually impaired, as well as the Lua language.

Brandon: Acquire experience with the usage of Lua as well as WoW add-ons.

Josh: To train himself with WoW & Lua.

Elliott: Contact the owners behind the website of www.blindwarcraft.com. The website was created to make WoW accessible for the visually impaired people. Also, become adept with the Lua language.

9/9/15

Members Attended

Trenton Balogh, Jehoshua Josue, Angelo Dejesus, Brandon Navarro, Elliott Ploutz

Progress Reports

All members submitted add ons for review by the team under the shared document labeled “Resources.”

All members created a “hello world” script in Lua.

www.wowprogramming.com was found to provide API functions and running Lua code in WoW.

Elliott & Josh contacted the webmaster behind blindwarcraft.com and established that he will be a tester for our program. May get in contact with other developers.

Goals for Next Meeting *(For Week 3)*

Current programming goals are to tackle the item aspect of the add-on. Two add-ons that currently exist which are under review by the team are “sellTrash” and “autoEquipQuestItem.” Our add-on will auto equip the best possible item and eliminate unnecessary items in the bag slots.

Trenton: Tasked with creation of design document. Edit introduction and executive summary.
Jehoshua: Will develop prototype images for portfolio 1 over the weekend. Will review code and design document.
Angelo: Tasked with creation of design document. Edit introduction and executive summary.
Brandon: Tasked with writing the introduction and executive summary of portfolio 1.
Elliott: Tasked with creation of design document. Edit introduction and executive summary.

9/16/15

Members Attended:

Trenton Balogh, Jehoshua Josue, Angelo Dejesus, Brandon Navarro, Elliott Ploutz

Progress Reports:

All members completed assigned tasks.

Goals for Next Meeting:

Just a reminder that the next portfolio due is 10/14, approximately 4 weeks from now. Graphical presentation of participation & accomplishments.
UI Menu for debug/test?

Trenton, Elliott: Develop code to interact with text to speech for bag functionality. Names subject to change.

Prototypes for:
coreParseCommand() - read chat commands
coreErrorOutput() - check for incorrect commands
bagModuleInit() - ready tables for bag stuff
bagScan() - read through and store inventory items

Jehoshua, Angelo, Brandon: Text to Speech addon, explore functionality, diagram of sorts. Presentation to other team members.

9/21/15

Members Attended:

Trenton Balogh, Jehoshua Josue, Angelo Dejesus, Brandon Navarro, Elliott Ploutz

Progress Reports:

Trenton: Trenton generated some test functions
Jehoshua: Found methods to adjust Jocys text to speech
Angelo: Found methods to adjust Jocys text to speech
Brandon: Found methods to adjust Jocys text to speech
Elliott: Clarified bag functionality.

Goals for Next Meeting:

Trenton: Make a slash command/event handler
Jehoshua: Further text to speech research
Angelo: Further text to speech research
Brandon: Generate simple functions
Elliott: For Elliott and Trenton -
Do we need a queue for items?
Can we get a description of the item?
Add bag initializer.

Ideas:
The current keys for spellbook, quest, inventory functionality can be mapped directly to our functionality. The normal functionality wouldn't be used.

9/30/15
Members Attended:
Trenton Balogh, Jehoshua Josue, Angelo Dejesus, Brandon Navarro, Elliott Ploutz

Progress Reports:

Trenton: Initial slash command handler completed. Hotkey '.' so that it ends the voice to text message.
Jehoshua: Able to get messages to the user from the API. Hotkey '.' so that it ends the voice to text message.
Angelo: Still experimenting.
Brandon: Wrote questing functions and separated files.
Elliott: Worked on inventory functions.

Goals for Next Meeting:

Trenton: Ownership and extension of spellbook functionality.
Jehoshua: Initial quest location equations for movement. (Specifically an equation for orientation).
Angelo: Keybinding items, spells, and interface. Help function for currently implemented functions.
Brandon: Quest functions, reading, iterating, etc. Collaborating with Josh.
Elliott: Have a working inventory reader. Get item information.

10/7/15

Members Attended:

Trenton Balogh, Jehoshua Josue, Angelo Dejesus, Brandon Navarro, Elliott Ploutz

Progress Reports:

Trenton: Done with Spellbook and awaiting keybind functionality.
Jehoshua: Ruled out some movement methods and developed equations.
Angelo: Completing Help command functionality and key binding functionality.
Brandon: Completing quest functionality.
Elliott: Completing bag functionality.

Goals for Next Meeting:

Trenton: Beginning combat functionality.
Jehoshua: Continue on movement functionality.
Angelo: Tinkering more with keybinding and get command line functionality.
Brandon: Finish quest functions and help with movement.
Elliott: Finish bag functions with money and keybindings. Help with combat.

Note, graph due in next portfolio.

10/14/15

Members Attended:

Trenton Balogh, Jehoshua Josue, Angelo Dejesus, Brandon Navarro, Elliott Ploutz

Progress Reports:

Trenton: Combat functions added.
Jehoshua: Attempting to triangulate position and find distances.
Angelo: Added more options to the help commands.
Brandon: Progress on questing but not complete.
Elliott: Progress on bags but not complete.

Goals for Next Meeting:

Trenton: Complete combat functionality.
Jehoshua: Continue with movement.
Angelo: More keybinds and help functions.
Brandon: Complete questing and move to combat.
Elliott: Complete bags and integrate all code.

We're all on portfolio duty with more to come especially in dates, interface design, and adjustment of UML's.

The team discussed the issue of clicking bodies to loot them. There is an item which blind players will most likely need: the loot-a-rang. The item does an area-of-effect (loot all surrounding options) and at range looting. This would solve many issues. In the future, we can contact blizzard about giving this item to advanced players, as this is a higher level item.

10/19/15
Members Attended:
Trenton Balogh, Jehoshua Josue, Angelo Dejesus, Brandon Navarro, Elliott Ploutz

Progress Reports:

Trenton: Not yet completed with combat utility but functional.
Jehoshua: Found distance conversion and wrote code for movement algorithm.
Angelo: Wrote keybind functions and help code.
Brandon: Not yet completed, but 3 more functions to add. Still in testing phase.
Elliott: 2 functions to add.

Goals for Next Meeting:

Trenton: Continue on combat. Maintenance and updates on code. *Portfolio: Update the combat, core, and spell book UML's/descriptions.*
Jehoshua: Add a function to determine if player movement is obstructed. Continue debugging movement algorithm. *Portfolio: Update the images. Add a description of the current movement algorithm.*
Angelo: Graphs for the portfolio, one for commits and lines edited. Beginning tutorials. *Portfolio: Update the meeting minutes. Update the Help UML and description.*
Brandon: Testing code and finishing quest functions. *Portfolio: Update the executive summary. Update the questing UML and description.*
Elliott: Testing and finishing inventory functions. Aiding in movement. *Portfolio: Overall editing and formatting. Update the inventory UML and description. Update the functional requirements.*

11/4/15
Members Attended:
Trenton Balogh, Jehoshua Josue, Angelo Dejesus, Brandon Navarro, Elliott Ploutz

Progress Reports:

Trenton, Jehoshua, Angelo, Brandon, Elliott: Completion of our presentation and to assess the opinions given during class. Which changes and whatever else to make.

Goals for Next Meeting:

Trenton: Code clean up.
Jehoshua: Assess Tomtom AddOn & code clean up.
Angelo: Code clean up.
Brandon: Code clean up.
Elliott: Code clean up, review presentation given our recording.

11/9/15
Members Attended:
Trenton Balogh, Jehoshua Josue, Angelo Dejesus, Brandon Navarro, Elliott Ploutz

Progress Reports:

Trenton: Updated functions.
Jehoshua: Updated movement functions with bearing.
Angelo: Updated help functions.
Brandon: Code clean up and quest functions.
Elliott: Added bag functionality.

Goals for Next Meeting:

Trenton: Continue with character equipment functionality with Elliott.
Jehoshua: Learn more about alternative movement algorithms like Tom-Tom.
Angelo: Update help functions based on code clean up.
Brandon: Add more functionality to quest functions.
Elliott: Continue with character equipment functionality with Trenton.

11/20/15
Members Attended:
Trenton Balogh, Jehoshua Josue, Angelo Dejesus, Brandon Navarro, Elliott Ploutz

Progress Reports:

Trenton: Character equipment may be out of scope due to time constraints, further discussion needed.
Jehoshua: Discussed with group about Tom Tom.
Angelo: Adjusted help utility to include when other utilities are on.
Brandon: Added more functionality to quest utility.
Elliott: Character equipment may be out of scope due to time constraints, further discussion needed.

Goals for Next Meeting:

Trenton: Portfolio work up and looking more into equipment panel.
Jehoshua: Portfolio touch up and code clean up.
Angelo: Portfolio touch up and code clean up.
Brandon: Portfolio touch up and code clean up.
Elliott: Portfolio work up and looking more into equipment panel.

11/25/15
Members Attended:
Trenton Balogh, Jehoshua Josue, Angelo Dejesus, Brandon Navarro, Elliott Ploutz

Progress Reports:

Trenton: Theorized on the character panel for the future. Worked on the portfolio.
Jehoshua: Adjusted movement function.
Angelo: Worked on the help function. Worked on the portfolio.
Brandon: Worked on the portfolio.
Elliott: Theorized on the character panel for the future.

Goals for Next Meeting:
Trenton: Practice presentation.
Jehoshua: Practice presentation.
Angelo: Practice presentation.
Brandon: Practice presentation.
Elliott: Practice presentation.