Accessibility modifications for mainstream computer games

Contents

Introduction

Problem

Providing accessibility consistently presents a difficult challenge for software developers. In software the limitations of input and output present many challenges to the impaired that must be overcome. While there has been great strides in providing accessible alternatives for software, in the now massive games industry there has been little progress.

"More than 40 years has passed since the first computer game was developed (referring to the general assumption that Spacewar from 1962 was the first computer game). Yet you still have the same implied prerequisites to play a game, i.e. full sight, hearing, cognitive, motoric et cetera functions. In short the game industry excludes many (or most) disabled, potential gamers. Compared with the web industry (only about 10 years old), the game industry has done very little (if anything) in the accessibility area." -IGDA[i]

Given how large this industry is, having recently outsold recorded music[ii] in profits and the estimated proportion of the populace that are impaired there is a lot of money to lose by excluding the demographic. If the highest sellers each year can be adapted from playable to unplayable and reach the same distribution of the market then it would easily pay for itself. For example the 2008 highest seller Mario Kart Wii, sold 13.7 million copies[iii], if it did not sell to anyone who was for instance severely visually impaired which represents 0.7% of the population[iv] then it stands that it lost 970,000 sales.

Aside from mere financial incentive there is also an argument that the current state of affairs for the disabled is unfair as they have few alternatives and are unable to participate in popular phenomenons.

The failing is due to the non-critical nature of games and the corresponding acceptance of losing a demographic and also the apparent difficulty of the task. Given little guides on the subject and the immense challenge of properly play testing with development it is unsurprising that there is little industry force. However, now is an excellent time for this to change, games no longer need a one size fits all scheme. All current generation consoles and gaming PCs allow for downloadable content and patches such that is has never been easier to retrofit a game to broaden the demographics who can play it.

Solution

What I am aiming to do with this project is attempt to discuss and justify through the development of software a method for implementing accessibility options in mainstream games. The original goal was to develop an example of software that could be implemented without any access to the source. This was explored in preparation, but for practical purposes I moved away instead into the development of a customizable integrated system. I hope to show this could be developed to a formulaic method and be known to be playable. I was specifically looking at low vision as defined by RT Jose.[v]

Context

Improving accessibility has always been an important task for software, the right to accessible alternatives has been enshrined in law since the Disability Discrimination Act [vi]of 1995. There have been numerous research groups set up in the field most attached to universities.[vii] Primarily they are looking at ways of improving software accessibility and much work has been achieved in providing

solid guidelines, hardware and assistive technology for a full range of disabilities.
In the game development industry there has been a limited amount of work. A number of papers have been written on the subject[viii]. Largely these have explored making games from scratch to appeal to disabled players or making specialist hardware for them. The former has had some success with a range of attempts at making audio games[ix] and simplified games for the disabled. I find however the question of integrating mainstream games more appealing given the impracticality of having separate industries for each disability. There has been many reported incidents of disabled individuals learning to play games based on very little information[x]. It seems that with a little help this could extend to reach a much larger amount of the community.
The impairment that I will be concentrating on will be visual impairment so its worth noting a little of the biology of the matter. Visual impairments are symptoms of a variety of eye disorders including cataracts, retinal degeneration, glaucoma, congenital disorders and infections[xi]. The term visual impairment covers a lot of specific symptoms including low visual acuity associated with Myopia or Hyperopia, as well as specific areas of lost vision such as tunnel vision and colour blindness. For this project I will be specifically looking at loss of visual acuity while discussing extensions in terms of conditions such as colour blindness. Colour blindness is the inability to distinguish certain colours, the most common red-green forms of colour blindness are protanopia and deuteranopia.

Preparation

OpenGL hooks and why I didn't use them
The original proposal for the project detailed the facilitation of the impairment modifications through the use of a hook to openGL. There were a number of benefits to this approach, not only would it hopefully allow me to demonstrate rendering modifications that would show what changes might help a low vision user it would also on completion be a genuinely useful tool. That is if it were to work for some subset of popular games, these games could be made more playable by the distribution and use of the hook. By moving the modifications outside the main code you could modify games you would normally have no access to and the hope was that a single modification set could be used for many games.
The difficulty of it however was apparent so it required a lot of exploration at preparation time. The method I explored was to use 'injection software' to replace the openGl.dll file with a modified version. This version awaited a call to getprocaddress, which was modified so that any other openGL command could be redirected to a dummy command before execution. This technique was developed by hacker sites that use these hackbases to cheat on online multiplayer games.
The aim was to modify all the calls to draw colour vertexes such as color3f, to modify the output of the image. I found a small demo for openGL and managed to inject the dll to change the colour of the output. This at least showed there was some merit behind the original idea.

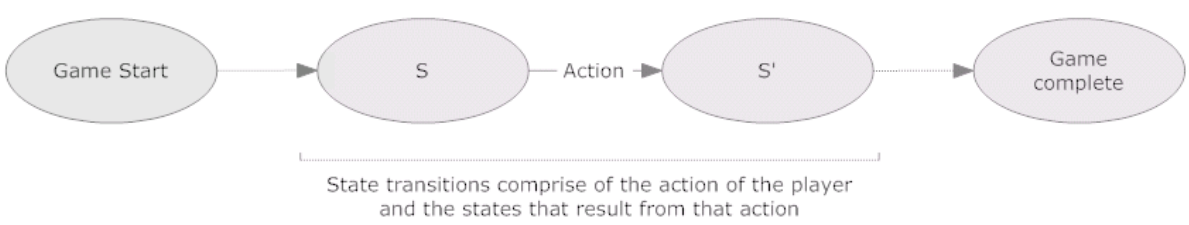
I had hoped to be able to do a pixel by pixel modification on the buffered image. This was possible however unsurprisingly it took many frames to accomplish and so could never be used at runtime on a real game.
Here we see the limitations of the method, only modifying the openGL commands is extremely constricting. The commands are all interfaces to a very exact form of rendering. We are given neither any information about the scene nor do we have a general description of the output (such as a pixel grid). This means the simple modifications targeted to make the desired changes are very difficult. Another consequence is that its very hard to make efficient modifications. The way to make changes efficient is either to call the algorithms once on load calls or to render them through the GPU which is able to deal efficiently with pixel modifications at runtime. If you change things at the openGL call point you add a massive inefficient overhead to the rendering. The most damning realisation however was unlike my original hope methods could not be simply taken from one game

to another, injecting the dll to the new software. Opengl is just an interface for a rendering engine and while it gives guidelines it is used vastly different from one engine to another. For instance one cannot assume that by changing all colour vertex points to green will make an image green as there are many alternatives to using colour vertexes to describe the model being rendered. A bigger problem and one I anticipated is that larger games become increasingly hard to tamper with, with specific checks in place to reject modified files as well as simply making certain assumptions about call times and dependencies that cause the injected dll to fail.
The combination of very restricted ability to modify images, the inefficiency at the point to do so, the lack of consistency of any changes made and the general resistance that most mainstream games had to modification caused me to abandon the openGl hook method. While I'm sure many of these would be possible to overcome, the resources and information for hacking games is sparse and the technical challenge grew into such a risky venture that for the main implementation I decided to move into direct open source modification of a game.

Minimum playable experience
Given the new limitation to the a single game I wanted to maintain some level of universality to the project, I wanted to show that the modifications I was to make could be applied to more than the single situation that I was applying. As mentioned in the introduction the limitation on development of accessibility features is not the difficulty of the change (which can be moderately easy) but the understanding of the effect. To this end I hope to explore a model for changing a game that can hopefully be applied to any situation.

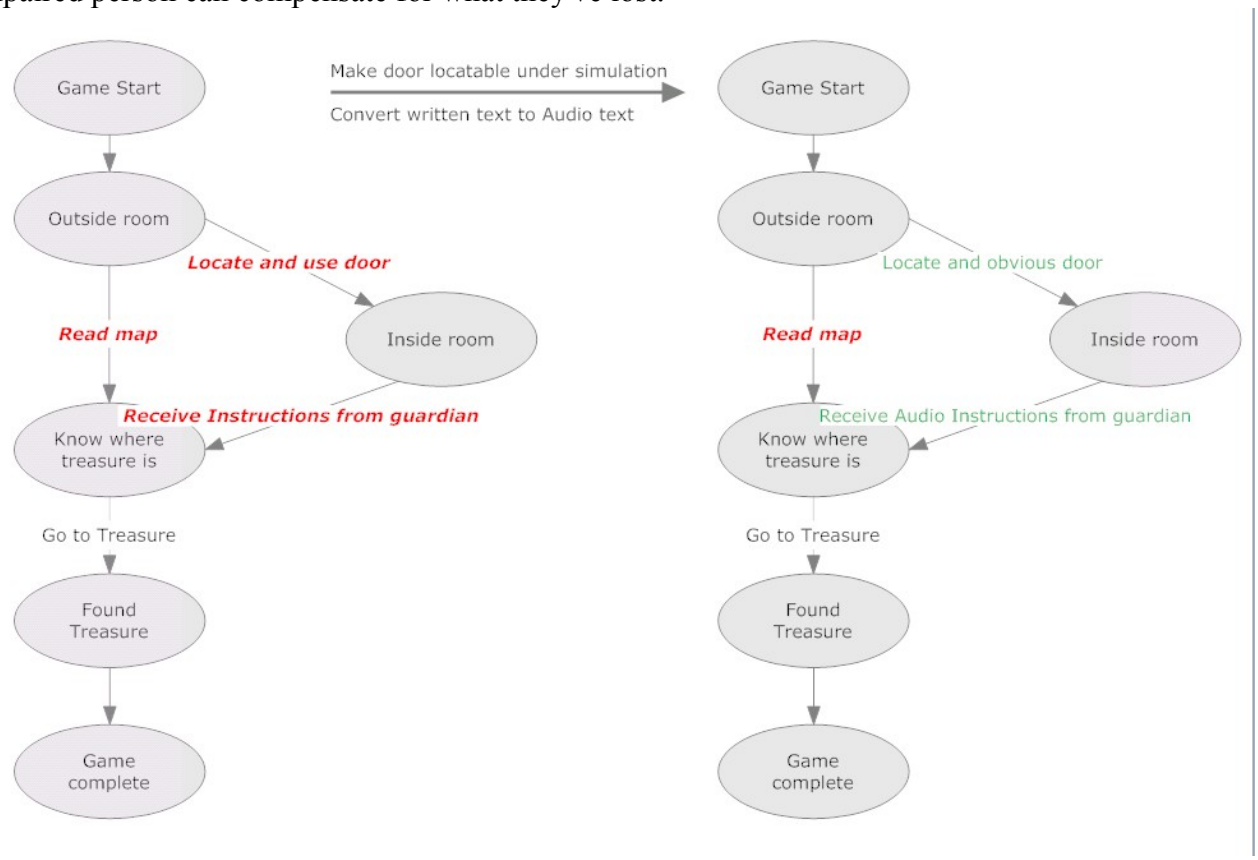
For the game to be considered accessible almost all judgements would be too subjective to allow for reasonable assertions so we require an objective definition. I will be using playable, that is one assumes there exists a state whereby enough of the content has been played to be considered a final state. For conventional games this would probably be when the credits roll, for less conventional games it may be that the designer must pick a suitable position whereby they believe the game has been completely played. Therefore what we are looking for is to achieve the minimum playable experience (MPE).
If we have this state we can say that if a player can get from the start state to the this state then they can play through the entire game and the game is playable.
This ignores fundamental aspects of game design, namely the most important that a game be fun. Obviously the fun aspect requires some instinct but for the application of this method unfortunately to keep it objective the fun concern is sidelined.

As we have a start state and an end state for our definition we have a standard AI goal finding problem. We want to find a single path from one state to the other where each action required to move along the path is possible in the game. To test it for a given disability we say that each action must be possible for someone with the impairment.



It was stated above that one principle difficulty is in finding a gameplay tester with the disability to

test at each stage, so it becomes necessary to simulate the impairment. This simulation must be at least as bad (that is as limiting) as the target impaired individuals impairment. If it is as limiting then we can say if every action can be completed under the simulation then the target individual is capable of completing them as well.
Obviously we expect a lot of the actions in the normal game to be impossible to complete for any severe disability hence the need to modify the game so there does exist a path to completion.
The advantages of using this graph based technique is that it helps avoid common pitfalls in accessibility design by being precise about what is being attempted to achieve.
For instance it is not necessary to provide the same experience to an impaired person as to the non-impaired. In fact it may often be impossible. What is necessary is to find some way in which an impaired person can compensate for what they've lost.

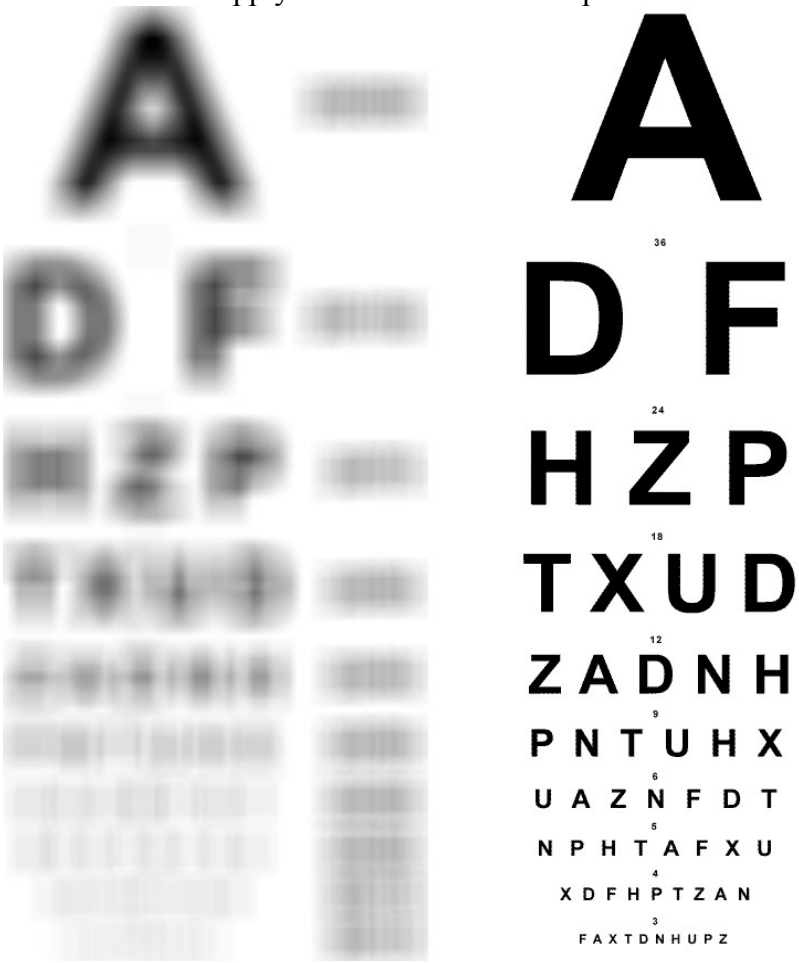


The most simple is in the cases such as Valve's Halflife2 series where closed caption subtitles were added in a patch to replace dialogue[xii]. These do not provide the same experience as sound but they do provide all the information so that the rest of the game is completely playable.
It also reminds developers to think outside the box. If one part of the game is impossible to play for a colour blind gamer then make a way of skipping it in colour blind mode, replace it or change it. As long as there is a path around the challenge then the game remains playable. Likewise if some part is too difficult make it easier for the disabled. There is a prevalent feeling that people need to design complicated technical workarounds to problems when as we can see it is quite possible to make a game remain playable (and fun) for an impaired individual if it is just easy enough for them.
There are of course drawbacks to this method, firstly if applied over zealously it can lead to the other common gripe in accessibility. 'Flying by bleeps'. If you make an entire interface for say a blind person to play a flight sim, where you use bleeps to show distance and position of objects. It may have all the information to play but lose the immersion of providing information in a realistic manner.
The other problem is to remember what is required for each action to be committed. Here we look to standard HCI design technique of cognitive walkthrough. For any action to be completed we must be able to infer a reason to why the player would know to do it, so while there may be some

areas where a disability doesn't force a player to stop it may make it so that the player would have no way of knowing what to do. After all taken to extremes if we forget to justify the players motive then any action at all is possible for perception disability.
It should be clear now that the other drawback to this method is the possibility of it ballooning into a huge task full of redundancy. To try and reduce this we create functions for the most common tasks. As is the way of most games these will normally be combat and exploration based skills. By lumping these up into context free functions we do not need to think about them every time, this is to help us divide the focus between specific problems such as a key that can't be seen against its background and general problems such as enemy monsters always being clearly visible. As always its a case of being over safe.

This method is aiming to guarantee a base level of playability to a game. Obviously to improve on that adding flexibility to the modifications is a must so that individuals can tailor the output to their needs.

Simulation of visual impairments
As stated above for this context the primary disability I'm looking at loss of visual acuity. Low visual acuity causes a loss of focus of a given magnitude at a given distance. To simulate this we use the standard optical tool of a Snellen Chart from there having decided on the level of loss of focus we apply our tool to make the correct line become unreadable. In this case the level was 20/70. We apply a Gaussian filter[xiii] which accurately represents loss of focus.
We then record the blur and apply this to simulate the impaired vision.



There is specific software for simulating colour blindness, this can be applied to still images with

selected forms of colour blindness. For common colour blindness such as red-green colour blindness we conflate the ambiguous colours into their composite, so in this case yellow. This allows us to see what can be distinguished and what can't be. It is also worth noting it is rarely the colour blindness alone that makes a game unplayable but the combination with visual acuity loss.

Testing a solution

In the testing of the project we care little about the safety critical nature of the code, thus test harnesses and more complex specification and verification techniques are dismissed. Instead what we care about is making sure that not only do the modifications work but that the method works as well. In doing this it will require a reasonable size group of testers who are willing to approach the unmodified and modified version of the game whilst having their version altered to simulate low visual acuity. While this may support the conjecture that the method worked it will be much better to also have a smaller group of testers who genuinely suffer low visual acuity so that the results apply to real people.

It will be important to both get quantitative feedback to justify the merits of improvement as well as qualitative feedback to determine how the player perceives the differences both in terms of enjoyment and functionality.

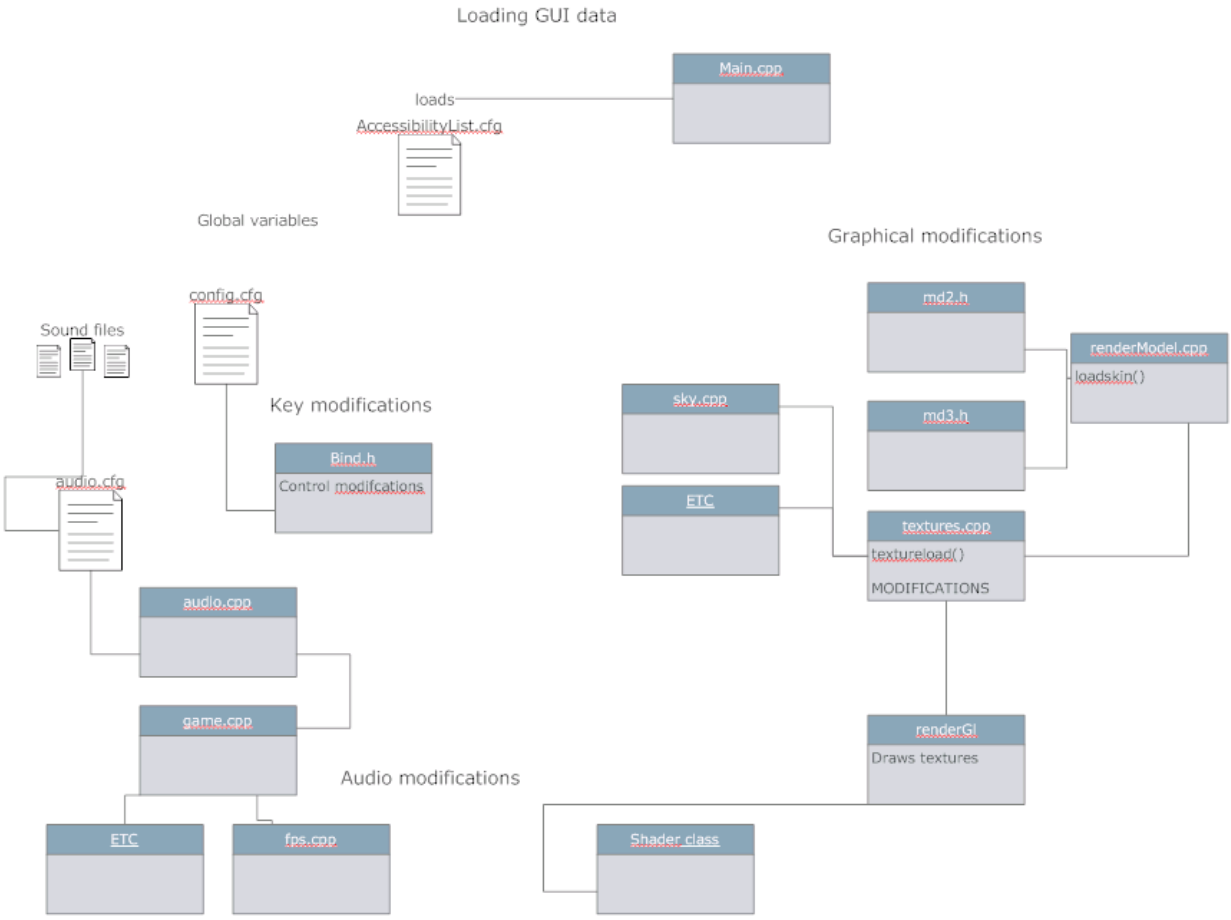
The source code

When choosing a specific source code to modify there were several specifications. I chose a first person shooter (FPS) as this genre sits comfortably on the boundary between that that is easily playable, where the focal point rarely changes such as racing or fighting games[xiv] and the completely unplayable such as real time strategy where extensive dialogue would always be a boundary.

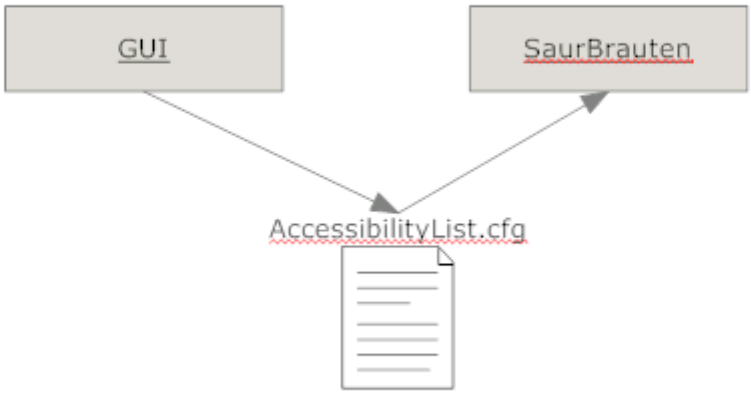
I therefore looked for an opensource extensive FPS and found a game called Saurbrauten also known as Cube 2.

Saurbrauten was developed by a group of German coders who were looking to create a large free game capable of being modified for a large online community.

As preparation it was necessary to come to terms with all the different complications of the was Saurbrauten was developed. Having little internal documentation, a lengthy period of trial and error modifying the source code was used to determine the following structure and implementation of the graphics engine.



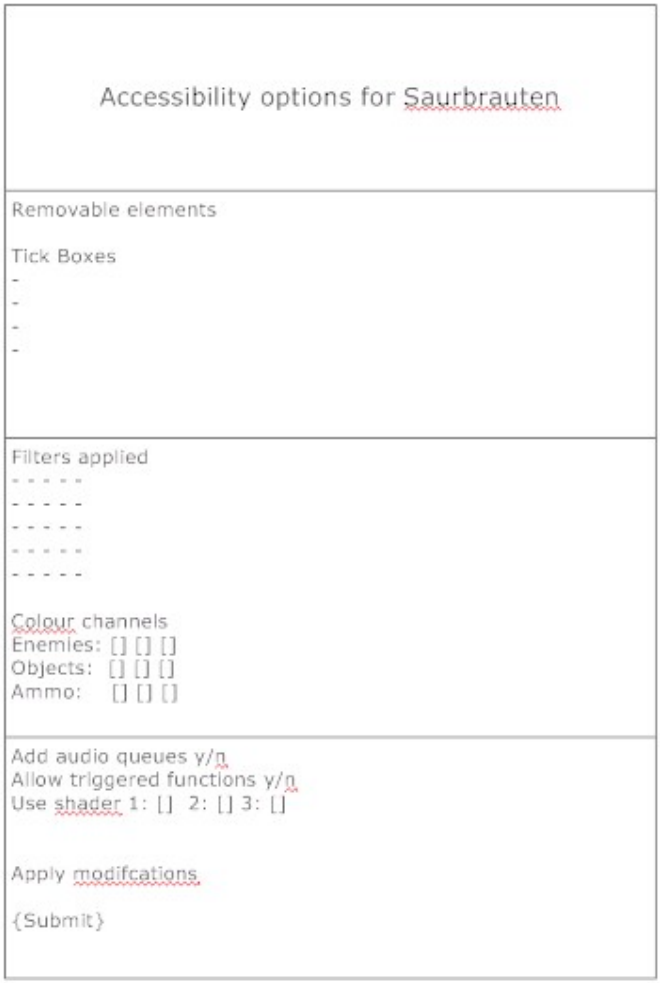
Divisions of workload
The final project will have two separate parts.
Connected as so



Given that I was working editing source code it was important to understand how my code would

work in relation to the above Saurbruaten modules.

I would need to create a Graphical interface so as to allow the different capabilities of the system to be easily edited, ideally without recompilation. I would do this using standard windows templates to take advantage of accessibility defaults in the windows system (so that it might be used by a partially sighted user) as well as to create a quick familiar layout. The display should be divided up by function and displayed so as to include all options in a clear manner. Something similar to the following:



The GUI is to create a config file that will hold the specs of the modifications to be used.
Then I would hijack the spec loading scheme to read my modification list and thus have a set of global variables defined.
Given the variety of locations and entwined nature of the modifications, their functions cannot be entirely separated into a different module. That is, given the different audio, texture, model, and post rendering nature of the modifications it does not make sense to group their function in a separate class structure instead it must be integrated into the correct place in the original Saurbrauten scheme.

The modification will be implemented in parallel with the above method for applying accessible modifications. The idea is that the scope of what can be changed is not directly related to what wants to be changed. With this in mind the modifications should provide a wide customizable range

for the modifications, as such every modification should be contingent on the requirement that it was requested in the GUI and any specific parameters should be also defined there. The secondary advantage to this is as mentioned above while not the focus of this project, maintaining the flexibility of the modifications is useful for people who wish to make the output specific to their requirements.

Requirements

Here is the list of modification capabilities that I intended to have to implement the MPE method.
The ability to edit textures and apply pixel by pixel modifications on them, specifically being able to apply filter convolutions to implement basic operations such as edge detection and sharpening of images. Other useful texture operations include changing the illumination, contrast and colour tints.
The ability to extend modality by adding in audio cues to convey information lost from the visuals.
The ability for the user to trigger aid to help them play, as an example the ability to magnify the screen.
Post rendering pixel shading implementing HLSL algorithms that would enable such effects as creating greater contrasts and edge detection.
Object colour tagging, the ability to add colour tags to distinguish various types of object in the world from one another.

Research

In preparation for this project it was necessary to investigate the following elements as well as the above.

The central relation between the understanding and the modification of images is the field of computer vision.

In working with Saurbrauten, I was forced to familiarize myself with C++ game engine design. The specific quirks of the language such as:
Structs, enumeration, header files, passing objects by reference and pointer arithmetic, all of which are used consistently.

Both Saurbrauten and more the development of the test of the openGL hook, were contingent on understanding openGL. OpenGL is a 3d graphics application programming interface for generating efficient high tech graphics.[xv]

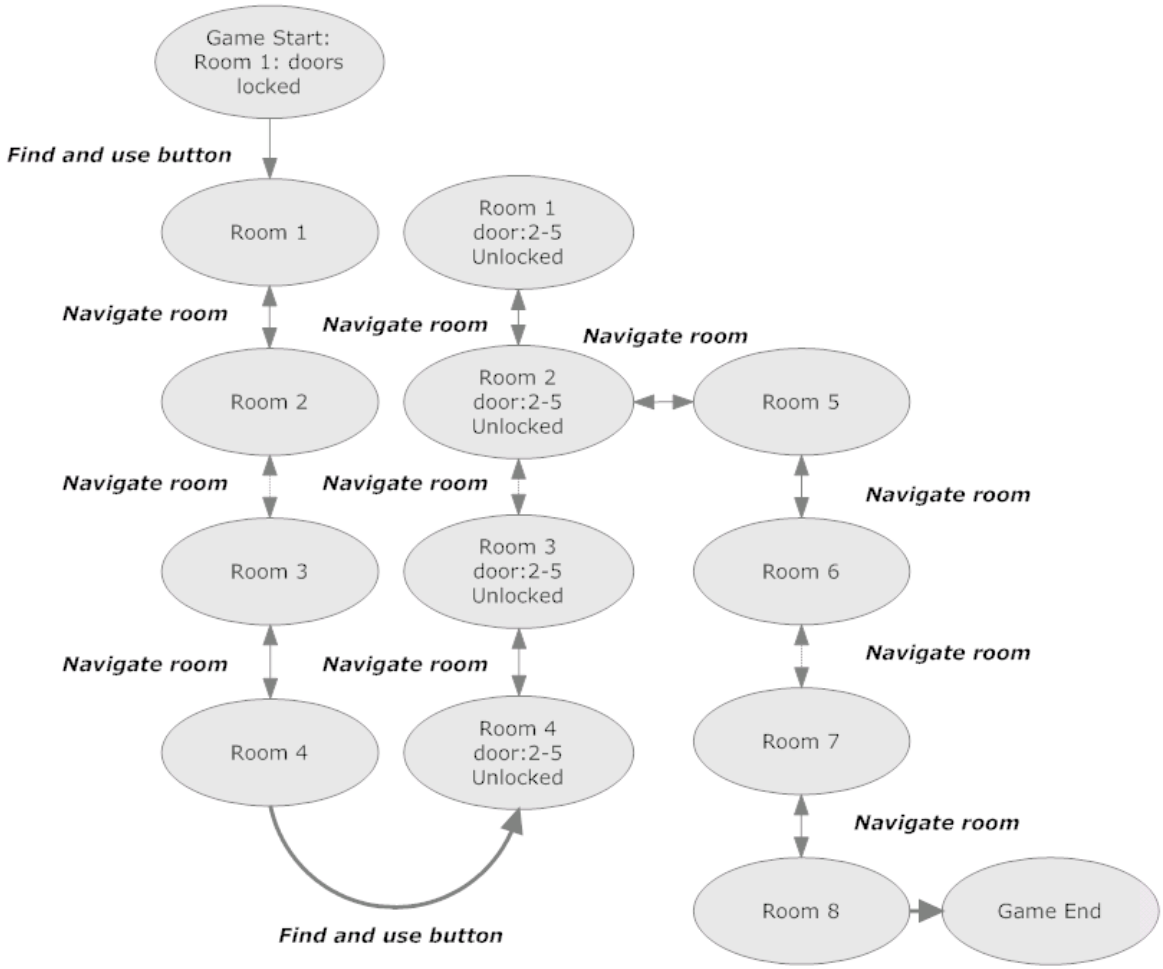
One interesting element about the textures used in Saurbrauten is that they are all SDL_surfaces, these are part of the SDL library of opensource graphics. They contain a variety of components for easy manipulation, however most important is the pixels element which is raw RGB image data.[xvi]

Implementation

Application of the method. (1000)

I have included the application of the MPE method because I believe it to be at the heart of this project in making it more than just the component pieces.
Firstly as the method is too lengthy to explain or test on a large scale I picked a small area of a level. Using the edit mode of the game I got this floor plan:

I have highlighted the rooms and numbered them 1-8.
The goal of the level is to get to room 8 in the upper right corner where a checkpoint is, starting from room1 in the centre where the spawn is.
This gives us our primarily state diagram.



The only in game requirement to finishing the level is the pressing of a trigger at 1 that opens the doorway from room 1 to 2 and another at 4 that opens 2-5.
The doors are 1-2, 2-3, 3-4, 2-5, 5-6, 6-7, 7-8.

Then elaborate each action into a set of requirements. It is at this point that we can generalize some skills so as not to repeat ourselves redundantly.
We lose some objectivity at this point as it is subjective as to whether these requirements are sufficient to complete the task, we are forced to envisage a typical player and their thought process.

We can navigate from room A to room B:

- If we have 'dealt' with threats in room A.
- Are able to determine the layout of room A, including walls, doors and obstructive objects.
- Are able to traverse the room to the door.
- Have justification for leaving the room via a door (mental model)
- Are able to use the door.

While it is not formalised that this is sufficient it stands that if one is in a room and has enough freedom of movement to navigate it (that is there is no threat) and one can determine the location of the door and has a reason to go through it. Then one can get to the door and if they can use it they will.

We can push a button in room A:

If we have 'dealt' with threats in room A.
Are able to determine the layout of room A, including walls, doors and obstructive objects including the location of the button.
Have justification to try and push the button.
Are able to traverse the room to the button.
Can activate the button.

At this point we are still too vague to test our requirements.
We can deal with threats if :

We can evade weapon fire and for all enemies in a room we can:
Locate each enemy
Correctly position ourselves to fire
Have justification to fire
Have the weapon capacity (ammo) to kill the enemy

That is to say if we are not killed ourselves, it is enough to be able to locate and shoot each enemy to kill all enemies.

To make our life easier we say that anything completely dissociated with the visual impairment disabilities can be ignored. That is the game is playable for unimpaired players.
So as a normal vision player must be able to conserve ammo so can a visually impaired player however this adds the requirement that the impaired player can determine the amount of ammo and distinguish ammo pickups. Likewise for manoeuvring we assume that the visually impaired player can too providing the layout of the room is determined, which is already a requirement. This just leaves the ability to see an enemy and its bullets and determine that it is an enemy. The latter is greatly helped by an ability to see that damage is being taken (which should lead to the concept of an enemy.)
So for combat we get the additional visual requirements of:

1. Always being able to distinguish each enemy from any background,
2. Being able to recognise the model as active and being able to observe damage being taken.
3. Being able to monitor ammo levels, being able to observe ammo pickups.

Applying a similar process to the navigation we get the requirements

4. To be able to distinguish the shape and layout of all the rooms,
5. For there to be enough distinction so as not to get lost (this gives justification for moving on from one room to the next)
6. And for the button that needs to be pressed to be obvious and visible.

Now we have our set of visual requirements that match our conditions for the actions we would like to be able to do, we can go about testing how well this works. First we do run throughs without visual impairment as a control test to validate that the level is completable with these visual abilities . As we go we take screen shots of pieces of evidence that we meet the requirements. Assuming the game is playable normally we then apply our simulation method to the screenshots and list any failings.
<For screenshots see appendix>
As we can see from the shots while we have full vision we can easily distinguish all the information but with the simulated low visual acuity we can at best guess about the concepts we are trying to

distinguish.
This sets us up with a list of problems that must be solved for the modifier developed below.

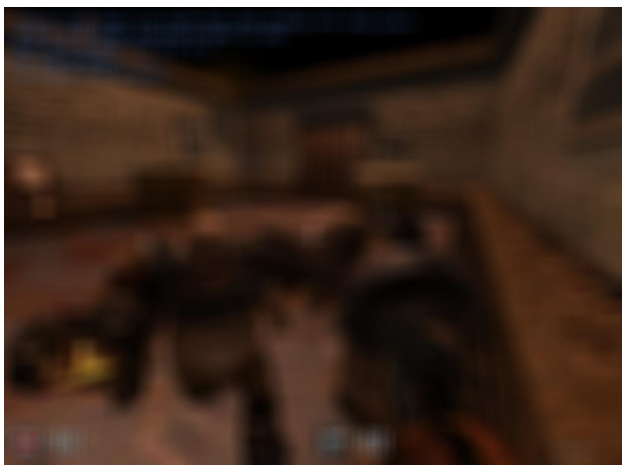
Given the above criteria I set about making a framework for adding modifications.
I installed the open source code and set it up to load from the visual studio solution.
Knowing that I was to write to a config file I created a function that loaded my file: Accessibility list on load and using the internal definition system I could define any variable from the file and use it on load.
For the small edits such as removing shadows or certain effects, it was just a matter of attaching conditionals statements based off which modification was ticked in the accessibility list. This proved slightly more interesting in the case of removing the sky texture. Interestingly if you simply remove the sky texture then a bug occurred where the area of the screen with literally zero polygon information instead of being black was the colour of the pixel at the last draw. It seems that this

depends on how the graphical processing unit buffers the screen as the results varied from card to card. Nevertheless to play it safe and avoid forcing the buffer to clear, if the sky is set to be removed the texture is set to black on load.

The bulk of the modifications fixated on the modifying of relevant textures. As explained above the textures are all stored as SDL_surfaces and all handled within the class textures.cpp. To avoid runtime overhead we want the textures to only be modified on load, so its at this point we create a function to handle the modifications. After establishing the format for pixel by pixel modifications of the textures I was able to begin experimenting with different types of graphical effect and see how they related to the above requirements.
So we run the game with the extraneous elements removed and apply the simulation remember we are looking to be able to extract information out about the layout of rooms and the location of enemies.

Still the blurred image without context is very hard to draw much information from but at least now we are not distracted trying to spot elements in the sky and we will not accidentally confuse an enemy with its shadow.

So looking at this image what can we say is the reason why we are not able to extract the necessary information. We would like to have a higher contrast in the image and ideally the edges to be more clearly defined.



To implement this I tried adding a contrast algorithm to be applied to each texture on load.
The contrast algorithm is adapted from the computer vision text. It calculates a pivot point by establishing the average illumination of the pixels, then for each pixel it multiplies through by the offset from that illumination, effectively increasing the contrast of pixel values around that point.

You can see in the non blurred image the details becomes far more noticeable, standing out clearly. The mottled floor compared to above is far more noticeable. However in the lower image we see this doesn't help much at all while certain patterns have become more distinctive the image is generally a mess and hard to extract any information from.

If we use a combination of the delta function (that which returns the original) and a Laplacian filter we get an edge enhancement filter. So to apply this we need to write a filter convolution algorithm. Here we take a matrix of known dimensions and apply to each pixel the surrounding component values according to the layout of the matrix. The algorithm is complexity $O(hwk^2)$ where h is the height of the pixel grid, w is the width and k is the length of the filter. Despite having 4 nesting levels of iterations it does not add too large a delay as long as k remains small. The filter algorithm gives us the ability to implement a range of filters that allow effects such as sharpening, blurring and edge detection. In this case it is an edge enhancement filter.
Use this with the image we will get textures with the edges highlighted. Such as here:

Again little to no help. Picking up the detail within textures doesn't help due to the blurring effect complete removing that detail, much the same things happen with other filters such as sharpen filters.

At this point saying that what we want to make really clear is the distinction between textures rather than inner texture details I attempted a much simpler modification, an algorithm that takes each pixel value and creates a new one which is a combination of a given ratio between its original value and the average value of the texture. If the ratio is one then the entire texture is replaced with a block colour.

This very simple modification helps somewhat in that because of the large patches of one tone, individual areas can be more easily identified in the post blurred state. However it certainly leaves a lot to be desired.

From here we progress to ideas about how to better distinguish one texture from another.
For instance the edge detection inside textures obviously didn't help. But would the somewhat blunt method of colouring the edge of textures in help us see where one texture ends and the next begins?
To do this all we do is turn any pixel within the specified number of lines from the edge black.
This creates the effect below:

The edges give a little help to seeing where one surface ends and another begins. Their real bonus is that by providing lines within a plane we get a quick impression of the angle of the plane making a wall much more distinct from a floor. This idea can be taken further by using the vertical component of the normal vector of the plane to modify the colour of the walls and floors. So for a floor plane the tag would be 1 and for a wall 0 and for slopes some value in between. This helps give a shape to the room that might otherwise be hard to distinguish.

These modifications help give us some sense of the shape of the world. As of yet though its still far too difficult to tell where an opponent is. A way round this might be to draw the models and the surfaces differently. For instance making one much brighter than the other, again this is very simple just an offset value added to the different textures. To detect when the texture is a skin of a model we call a separate function on loadskin that handles the different model values.

Above we see the models made dark against the bright background. This makes the models clear and the brightening of the surfaces helps make the shape of the room clear. However we have the problem that the enemy looks very similar to a crate. While it is important to see both it is very important to tell between them. The problem with using brightness as a factor to pick out significance is that the scale varies heavily depending on the lighting in the room. Meaning that it certainly doesn't provide a great device for picking out a range of different brightness tagged objects.
Colour is one alternative here as it does provide a range such that a clear semantic range can be established.

Here I've used 3 tags I've increased the red element for enemies, blue for objects in the scenery and green for ammo, given the generality in the original Saurbrauten code the only way to differentiate the skins of models was to do text string comparisons between the names of the relevant monster types and the skins being loaded. This allows a case switch as to which colour tag the model gets. The colour tagging is just modifying each of the RGB components of the pixel by the a given factor.

You can see, despite not being able to tell by the shape of the object, what it is instantly. You can spot the red enemies, the green ammo and the blue scenery. This form of semantic help gives a huge advantage to the player but has the consequences of removing much of the immersion. The colours are no longer are that of a real world. The question of whether this is an acceptable loss is very much down to the developer.

Here's the shot again with some of the other modifications added to it.

By removing information that distracts from the gameplay information and adding we begin to see a much clearer idea of the world and how to act within it.

This has helped with many of the requirements but we still would like information about health and ammo so as to allow for better performance at the basic combat elements. The entire UI is clearly lost as the simulation shows. There are very few ways in which onscreen text can be made readable for the partially sighted, instead it is best to change modality and convert visual information into audio information.
This is relatively simple given the infrastructure already in place. Sounds must be created then added to the package folder, referenced in a config file, enumerated and then are available anywhere the sounds header is included. I added a variety of sounds in to help convey otherwise lost information conditional on the modifications being switched on. As it would require too large a volume of sound files to have a voice read out every health value (of which its possible to range up to several hundred) I instead activated sound clips on thresholds, so that when health passes certain critical levels the player is reminded. I also used sound to notify the player about the gun that was held, the amount of ammo and the type of pickups. Information that would otherwise be lost.
Perhaps most importantly sound clips are used to help direct players, sounds are triggered to notify locked doors and when they're opened. I investigated using sound to guide the players by setting up a sound sample to activate from a point source, so that the sound was louder the nearer the player

got to the point. However I found it confusing and detracting from the game play to sue beacons in this way.

Key commands (zoom)

Adding screen magnification ended up being very easy. The game already had a zoom function that could be set to be always available and by modifying the speed and distance you zoomed and allocating it to an easily accessible key it meant there was always available a magnification.

I investigated the use of pixel shaders to do post render algorithms to modify the output. These help as the given amount of contrast required from different positions to distinguish different elements varies. The problem of course with only changing the textures involved is that any inter-texture relationship and any general information obscured by the rendering technique cannot be modified for. I attempted to implement a contrast shader as an example of this technique. The problem with pixel shaders is as its all computed on the GPU in parallel is any algorithm must treat the pixels as isolated objects, so the contrast algorithm can only increase contrast around the central grey value. To implement a shader one interrupts the openGL pixel render algorithm and replaces it with ones own this means that a shader must implement not only its own algorithm but all those of what is replaced. The shader that I was looking to incorporate were coded using HLSL, a shader language. Unfortunately due to time constraints they were never fully implemented in the project.
Instead to handle bad lighting situations a trigger based off the modifications was set to give a more basic uniform lighting and thus avoid dark areas.

With the set of modifications completed I added a GUI to allow easier defining of the modifications . I implemented the windows .net standard appearance. The GUI is the most basic type of form. On submission it writes out its values to the Accessibility list config file in the appropriate format. When making it, the only interesting complication was trying to make sure that it would be theoretically navigable by a visually impaired user. Accessibility hardware is designed to work for this specific format. As the developer it is my job to make sure that the correct accessibility tags are in place and that for instance focus changes happen in a logical order as accessibility aids run off these factors rather than the visual layout of the input boxes.

Lets return to our previously failed requirements

1. Distinguish each enemy from background: Is clear when enemy is dark red and the backgrounds are all light non-red colour.
2. Being able to recognise the model as active and being able to observe damage being taken: The models are active when moving, they can be examined further using the zoom magnification. Damage is indicated with audio signals.
3. Being able to monitor ammo levels, being able to observe ammo pickups: Ammo pickups are a dark blue against non-blue backgrounds. Ammo levels are reported with audio cues. Getting pick ups is also described using audio cues.
4. To be able to distinguish the shape and layout of all the rooms: The rooms have flat averaged light coloured walls with clear black lines separating planes and showing their orientation. The effects of applying the colour filter exaggerates the differences between walls and floors already present. Door audio cues also may help.
5. For there to be enough distinction so as not to get lost. Nothing specific. Hopefully increased understanding of surroundings will help.
6. And for the button that needs to be pressed to be obvious and visible. Buttons tagged dark green against light backgrounds. Clear audio cue on press.

Evaluation (2000)

The requirements of the method have been met but this tells us little about the merit of the method itself until we can compare this to real user data. As with the very premise of this project visually impaired test data is relatively hard to come by. So the principle contributors would have to have their eyesight artificially distorted. This was done a number of ways to create a range of simulations. The closest to genuine visual impairment was the correctable myopic testers who would simply play the game without their lenses correcting their vision. To bolster these numbers and to provide a more extreme form of visual distortion fully sighted testers were also asked to contribute whilst wearing glasses that were out of focus for them and further obscured using a thin smeared layer of liquid.
Each tester was asked to test their vision on a Snellen chart so as to give a value for the loss of visual acuity in the case of those wearing obscuring glasses I attempted to apply a layer such that the vision was obscured to the 20/70 level that I had been using as for simulation.
Each tester was asked to tell the difference between an image and the image with the Gaussian blur applied, many stated that they could not tell them apart. This was used as a justification that the testing impairment was at least as restricting as the image simulating impairment.

Each tester was asked to play two run throughs of the level. While doing so they were asked to comment on their actions and their reasons for doing so. They were also timed and had their number of deaths reported. There is a disadvantage of attempting to extract both verbal and performance data as one might well effect the other. However given the obvious limitations in getting large amounts of user testing and the incredible usefulness of both forms of information I thought it was worthwhile.
One run through would be with modifications applied the other would be without. For each test this order was reversed as obviously one expects the second run through to be far better than the first.

No matter how much data is collected from these various forms of simulation the data that we would really like is that of genuinely low vision players. To do this required a further search then within the local campus so I resorted to the Internet. As I did not feel comfortable asking these strangers to install the entire game and modifications on their computers to allow them to test themselves and also due to the numbers being so small that any data would not be significant I resorted to a simpler approach. I sent around a copy of video of the modified and non-modified versions being played and asked for comments on what could be distinguished in the video. While not completely accurate as much of the methods for acquiring knowledge about ones surroundings in a game comes in reaction to how you're moving however I hoped that it would still shed light on the underlying question of with the modifications achieved the requirements.

See appendix for statistical results.

As well as the performance data I collected the verbal comments and then collated them into general points. Here are the rankings.

Some comment on the significant comments.

Regular version: Player did not realise they had died and wondered where they were at respawn. This was presumably due to no information being communicated about health as well as the messages about death being missed. The confusion shows that players are playing using largely relative positions. Remembering how far they have moved rather than looking at their surroundings.

Both versions: Could not distinguish between enemy and enemy corpses. The player identifies the location of an enemy through incoming fire and in the modified version by distinguishing it, however no details are determined so the difference between dead and alive is only determined by behaviour. (This would be corrected through simply removing the corpse.)

Regular version: Could not see brown enemies. The brown enemies were largely reported to be invisible without the modifications. This seems like it was one of the highest contributors to deaths and frustrations. This highlights the fact that it is often just the small coincidences that can ruin gameplay for the impaired. The modified version did not suffer this problem.

Actual low vision feedback.
Here is the feedback from the low vision testers: Awaiting

As an extension to looking at loss of visual acuity I was interested in how one might repeat the method with colour blindness. This was a low priority so I could not facilitate user testing but I did run the colour blind simulating software on some images for comparison. One can see that by simplifying the image we gain a small amount of clarity but what we lose when we lose colour is the distinction between some of the colour channels. For instance in the first image we find the monster no longer as clear against his background while the blue crate shows up easily. This highlights the advantage of having a customizable product so that the right colour tinting can be chosen for the eye condition. It is most likely that colour blindness alone will not require modification however many people who have low visual acuity also suffer colour blindness so where there is overlap to get the best results there must be flexibility.

There is a series of factors to question about the approach that I took if it were to be applied to a commercial game: performance overhead, practicality and success rate.
The performance cost for these modifications were extremely low. Running the additional algorithms at load time increased the load times by the following:

These could easily be optimised for a more complex project.
Meanwhile runtime performance took no hit. In fact one might estimate that if there was to be a framerate drop there would be a better performance in the modified version as it has less information to convey.
More generally for accessibility modification using this method there is no restrictions that would imply the use of high overhead algorithms. Had the pixel shaders been implemented they may well have caused a slow down in Frames per second. The advantage of the method is that if there is a workaround to not doing this it is available to find (as opposed to attempting to make a ubiquitous solution).

The practicality is directly related to the amount of modifications which as I just said is as flexible in this method as anywhere. The other note to practicality is whether the method itself creates an excessive overhead. I would argue that if the designer is working from good design docs they should be able to break down a set of basic skills and requirements for a level in a few hours, the merit of this is directly related to the successfulness of the method.

The modifications were successful but it is very hard to gauge whether or not they would have been as successful if the requirements had of been chosen by eye rather than as a formation of the tasks of the level. There were many advantageous modifications that were missed because they were not identified as necessary. This is the negative aspect of the method in that by only attempting the small goal of being playable we don't strive towards doing everything that we can. What we can say of the method is that it did achieve its aims of producing a playable experience.

Conclusion (1000)

The outcome of the project is that there is little corroborative evidence for the method itself being any better than a less formal approach. In fact from the lessons learnt during evaluation there can be seen many other possible approaches that might bring better playability.
For instance the user testing was very rewarding in providing feedback. One approach might be to integrate user testing into the development of modifications far more often so as to gain feedback about where further modifications need to be made. For a mainstream studio it is obviously possible to have QA testers do this work under the simulation that I used however is the best information comes form low vision users it might be best to consult these as often as possible. This can be difficult given the fear of redistribution of games that are sent to remote locations.

The modifications themselves while broadly a success provided a few lessons. Firstly the simplest modifications tended to have the best results. Complex edge detection provided far too much edges in all the wrong places where as just drawing lines on the end of textures in this case created the effect I was looking for. The advantage of this is that for low visual acuity it would seem that a lot can easily be done. The advantage of keeping it customizable is to cater for the very individual nature of visual impairment. Given the simplicity of the modifications combined with customizability gives a very practical way of providing accessibility for a wide range of different visual impairments.

If I were to repeat the project I would have the knowledge that the original task of creating the openGL hook was not going to work. With this in mind instead of dividing my preparation over these two separate projects I could have focused directly on the modifications and had more success with the developing of pixel shaders and a wider range of visual impairments.

The purpose of this project was to investigate accessibility in games and while as an undergraduate project I doubt it will have any effect I do hope that the games industry makes progress in opening up games to include all markets. The industry is growing rapidly and will have to face these barriers eventually. The most advantages will probably come from new hardware that can act as an interface for impaired users however I feel there will always be instances where indirect manipulation causes a conflict and here it will be necessary to provide modified versions.

i International Game developers Association Accessibility in Games: Motivation and Approaches: http://www.igda.org/accessibility/IGDA_Accessibility_WhitePaper.pdf

ii Entertainment Retailers' Association, UK Market Statistics, 2008
http://www.eraltd.org/_attachments/Resources/yearbook.pdf

iii Nintendo Financial Results Briefing for Fiscal Year Ended March 2009: http://www.nintendo.co.jp/ir/pdf/2009/090508e.pdf#page=6

iv Page 1
Statistics on Vision Impairment: A Resource Manual April 200 2,Robin Leonard
Arlene R. Gordon Research Institute of
Lighthouse International
http://www.gesta.org/estudos/statistics0402.pdf

v R.T.Jose, Understanding Low vision: http://jvi.sagepub.com/cgi/pdf_extract/2/3/94

vi http://www.opsi.gov.uk/acts/acts1995/ukpga_19950050_en_1

vii ARC: Accessibilty Research Centre: http://rime.tees.ac.uk/arc/aboutus.php
digital media access group: http://www.dmag.org.uk/
ARG: Accessibility Research Group
University College London: http://www2.cege.ucl.ac.uk/cts/arg/

viii Computer games and visually impaired people
D Archambault, R Ossmann, T Gaudy, K Miesenberger - Upgrade, 2007:
http://cedric.cnam.fr/PUBLIS/RC1204.pdf

Tactile Interactive Multimedia computer games for blind and visually impaired
D Archambault, D Burger, S Sable

ix http://www.audiogames.net/

x http://www.wired.com/gaming/gamingreviews/news/2005/07/68333 Blind Mortal Kombat player

xi Low vision, Principles and Practice, Christine Dickinson

xiiGame accessibility article on closed captions in games
http://www.accessibility.nl/games/index.php?pagefile=auditory

xiii A Gaussian filter creates a Gaussian blur, a smooth blur that removes information. In terms of representing lack of focus one might prefer Bokeh effects which simulate the lack of focus better however here we just want the simplest method for reducing infromation.
http://en.wikipedia.org/wiki/Gaussian_blur

xivArticle on personal experience with a collection of game genres. http://www.game-accessibility.com/forum/viewtopic.php?id=243

xv OpenGl official website, http://www.opengl.org/about/overview/

xvi SDL tutorial, http://www.linuxjournal.com/article/4401