# Crafting ConFlo

A new approach to dialogue systems in adventure games

Written by **Tim Hengeveld**
Game Design & Development
2010/2011

Supervisor: **Corné van Delft**

**TABLE OF CONTENTS**



**SUMMARY**

Hengeveld, T., 2011, *Crafting ConFlo: a new approach to dialogue systems in adventure games,* Hilversum, Utrecht School of the Arts

While being adequate within the context of gameplay, most dialogue systems do a bad job of accurately representing all the nuances of real-world conversations. Therefore this thesis examines the current state of dialogue systems in videogames and charts their strengths and weaknesses in order to find ways in which to **develop a new kind of system that grants players a more natural and direct control over dialogues**. The main focus here is on the way in which the dialogue is controlled, disregarding other factors such as body language (animations), intonation (voice acting) and the like.

To achieve this goal, the author performs a careful analysis and categorization of the different types of dialogue systems currently in existence, coupled with an examination of the field of linguistics in order to ascertain relevant characteristics of real-world conversations that may be used in the creation of a new dialogue system. These findings are then used to develop several prototypes.

Upon examining contemporary dialogue systems it becomes clear that most of them are focused on acting, not reacting. The player is always the one asking questions, and when he receives the answers the dialogue is done. Comparing this to real-world conversations revealed a large gap where other characters could participate in the conversation and the player could respond to what they say.

The author then conceptualizes five different prototypes for dialogue systems that incorporate these findings. However these do not represent a final solution for creating a more natural and realistic dialogue system, merely a step forward. There are many other factors to be considered that fall outside the scope of this thesis, but there is one thing that becomes very clear: linguistics can be a powerful tool in creating a more natural and realistic dialogue system and therefore it warrants further investigation by the games industry.

## 1. WHAT IS A DIALOGUE SYSTEM AND WHY DO WE NEED IT?

As humans, we talk. Some more than others, but we talk. Or we write, so others can read it later. It is our main method of communication. We use it to ask for information, tell stories or simply connect with other people. It's an integral part of being human.

And so it happens that in games we have to talk too. In some more than others, but we talk. To ask for information or… well mostly to ask for information that furthers the plot. It is rarely we talk for anything else in videogames. But more on that later.

And to talk with these virtual characters we need an interface. Speech recognition is getting pretty good these last few years, but it is far from being at the point where it can perfectly parse everything we say, in any language. So we have to make due with *dialogue systems*.

These systems come in many shapes; from plain lists of sentences to prompts where we have to enter the text ourselves (with middling success) to colored shapes corresponding to a topic; we have our bases covered. But these dialogue systems have always been built in service of the *gameplay*, not in service of *communication*. And here we stumble upon our first and foremost problem, described in the next chapter.

## 2. THE FUNDAMENTAL DIFFERENCE BETWEEN REAL CONVERSATIONS AND GAME CONVERSATIONS

There is unfortunately a problem that in my eyes plagues the lion's share of game dialogues, especially those in adventure games: there is a lack of natural interplay between characters.

That is because **game conversations** usually have a single goal: *giving the player more information about the game and the current objective*. This often results in something more akin to an interrogation than a conversation. The player rattles off a list of inquiries that the NPC[1] answers with a small monologue. Some responses may be very well written, but they still offer very little opportunity for player input.

**Real conversations**, on the other hand, are largely about *strengthening bonds between people*. They are cooperative instead of competitive. Each speaker contributes his own part to the conversation, and there is no single person who constantly badgers someone with questions and then leaves - generally speaking. People use conversations to get to know eachother, an activity which, aside from the adventure game genre, we rarely find in videogames – unless it is to rack up enough friendship points to unlock a roll in the hay with that witch from *Dragon Age* (2009).

So could there be a way to make game conversations feel more like real conversations, without nescessarily sacrificing the primary focus, which is gathering information?

To try and achieve this goal, I have formulated the following **research question:**

*How can we give the player a system for more natural and direct control of game conversations (while keeping extra work required from the developer over traditional dialogue systems to a minimum)?*

I put the last part in parentheses because I think minimizing extra work is an important guideline in building a new system, but at the same time I do not want to discard a good idea with a lot of potential solely on the fact that it would increase the developer's workload.

I will try to find an answer to this query via these **subquestions:**

- What are the characteristics of a typical conversation?
- Which elements thereof are suitable for adaption into videogames, and which aren't?
- What has been done before in this field, and which lessons can be learned from that?

I will start with the last subquestion, which in essence is a *repetoire analysis*, in the next chapter.

And to clarify my terminology in this thesis, when I use the word **conversation** I mean the actual spoken exchange between two or more characters, and the word **dialogue** represents the virtual 'container' that holds the entire conversation, including the interface and code behind it.

[1] Non-Playable character: a character that is not controlled by the player

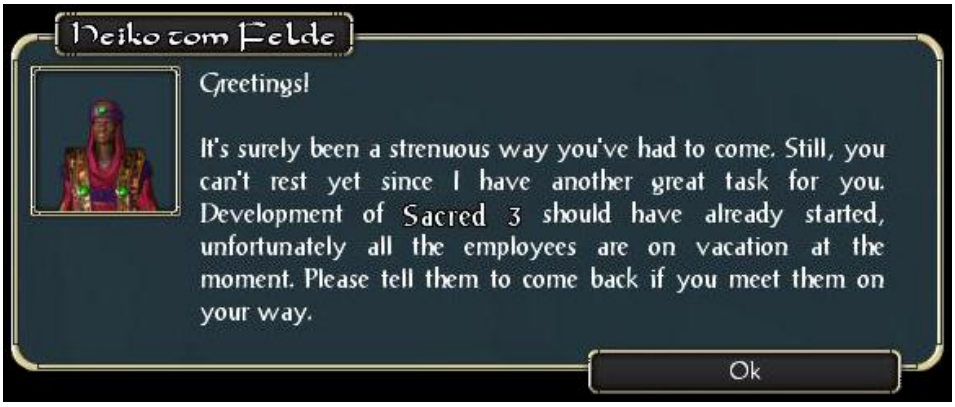
## 3. EXISTING DIALOGUE SYSTEMS

To classify the most common variations of dialogue systems in videogames, I will base my categories on the ones coined by Brent Ellison in his article *Defining Dialogue Systems* (2008) on Gamasutra.

### Non-branching dialogue

This is the simplest form of dialogue. Players can approach an NPC and initiate a conversation that usually consists of only a few lines of text. What the NPC has to say may differ depending on certain game conditions (such as if a quest is completed or not), or it may be a random filler line to give the player some feedback when he tries to communicate with an NPC.

This type of dialogue is the easiest to create because there is no player input and thus it is entirely linear. Therefore its use is quite common, especially among RPG's[2] where the player would find a lot of townfolk that are not particularly related to what is going on in the story.




### Branching dialogue

Branching is a big step up on the dialogue ladder. All of a sudden the player is given influence over the direction of the conversation. They can choose from a number of options that take the conversation in a different direction, something influencing how the NPC feels about them.

The way in which these options are presented can vary wildly between games. Some games choose to print the complete line as it will be spoken by the player character, others choose to shorten these options into keywords, and others still represent the tone of the option with a symbol or a color.

Under the banner of branching dialogue we can identify a few different variations:

### Branching dialogue: Multiple-choice

The most common manifestation of branching dialogue is *multiple-choice*. Players now have input in the conversation by selecting a response from a list of options. A classic example of multiple-choice dialogue is ***The Secret of Monkey Island*** (1990), an adventure game about a wannabe pirate.

[2] Role-Playing Game: a game with a strong focus on character development


Players click on an NPC, a list of questions pops up and the player works through them, usually to gather information about his current objective. Sometimes there are multiple ways to ask the same question, which sounds like it has an impact on the direction of the narrative, but more often than not it is just a way to pad dialogue with some extra options to liven up the game, only giving the player character a different outward appearance without actually affecting anything. Often these slightly differing options have a tendency to all loop back into the same outcome.

This is mostly because true branching dialogue eats up a lot of development time, all the different paths to keep track of and different outcomes to write, animate and possibly voiceact. There are only a handful of developers that can pull off such a feat, such as *BioWare* with their *Mass Effect* and *Dragon Age* series, or *Quantic Dream* with ***Heavy Rain*** (2010), an interactive drama.


As a console game, *Heavy Rain* shakes things up a bit by binding dialogue options to the controller buttons. They are also represented as unseen physical objects in the game world, swirling around the player character. This also helps to illustrate the mental state of the character as the options shake, spin faster and become blurry when the character is nervous/scared/angry. And depending on which option you choose you actually alter the direction of the narrative, leading to one of several different possible endings. A lot of work, but very beneficial for the player's immersion and sense of freedom.

*Heavy Rain* is also to my knowledge the only game that effectively mixes dialogue in with regular gameplay. While the options are visible you can still walk around and interact with the environment, a pleasant development that illustrates one of the areas of improvement for dialogue engines I will propose in chapter 4.

**Branching dialogue: Hub-and-Spokes**

While not differing that radically from multiple-choice, the *hub-and-spokes* method takes a different approach to structuring options in a branching dialogue. Instead of presenting the player with a list of options for the entire conversation, the dialogue is broken up into 'spokes', roughly corresponding to

different topics, that can be accessed from a central 'hub'. Entering a spoke opens a new hub with subtopics that can be explored before returning to the main hub.

The clearest example of this method is ***Mass Effect*** (2007), a scifi action RPG.

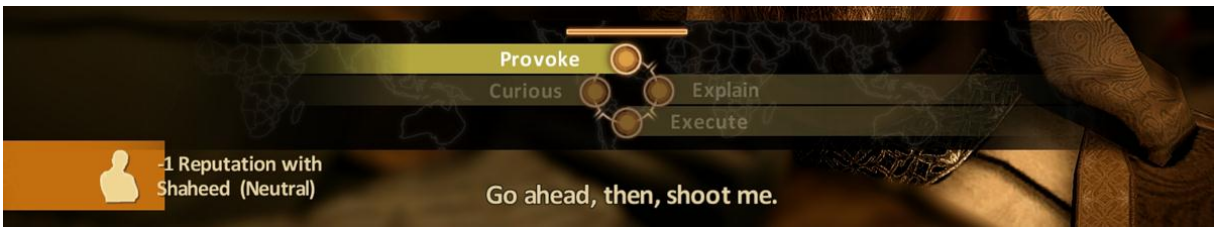

Its implementation of hub-and-spoke dialogues give players more freedom to explore the conversation in any order they wish. Though this could lead to the player repeatedly encountering the same lines of dialogue as they transition in and out of the different hubs.

Another advantage *Mass Effect* has over other dialogue systems is that it presents players with the dialogue options before the NPC finishes speaking. This way players can already select their next response before the conversation falls silent. However there is no incentive to do so, as the NPC's seem to have infinite patience and stand around calmly waiting for the player to ask a question. Aside from this being useful for contemplating difficult choices to be made, leaving the conversation idle for too long could lead to a break in immersion.

A game that tries to remedy this is *Alpha Protocol*.

**Reactionary dialogue**

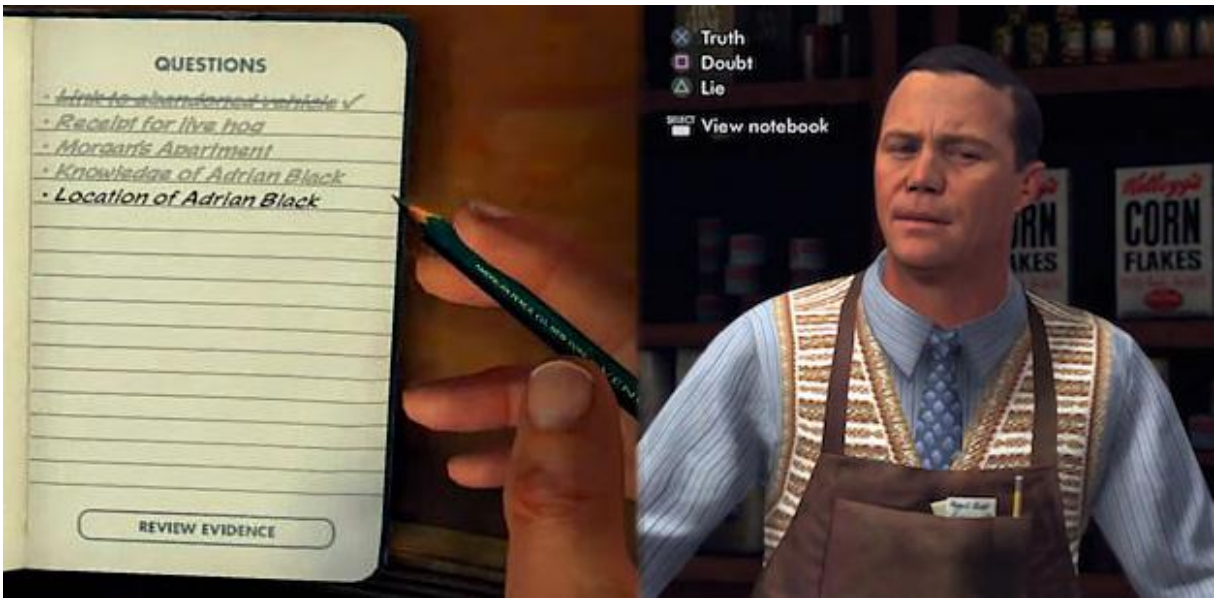
Reactionary dialogue is a category I defined myself to house (so far) two games, because I feel they do not fit within any of the other categories. The first one is ***Alpha Protocol*** (2010), an espionage RPG.




*AP* shares similarities with branching dialogue, but is different because it is focused more on *reacting* to characters instead of barraging them with questions. In *AP* you mostly choose **stances** instead of topics/keywords. Each NPC has a stance they favor, such as *professional* or *suave*. Choosing the stance matching the NPC's predisposition will improve how they feel about you and vice versa.

And apart from presenting the options during the conversation like *Mass Effect* does, *AP* also has a timer that starts running as soon as the player is presented with the dialogue options. This timer runs out at the same time as the NPC stops speaking, thereby keeping the dialogue flowing at all times. This does put extra stress on the player, as they have to make a decision in a split second, something that takes some getting used to. I personally found it enjoyable but I can see how people would also want some time to contemplate difficult decisions.

The second game I've filed in this category is ***LA Noire*** (2011), a film noir detective epic.




On the surface the dialogue system in *LA Noire* seems like a traditional branching dialogue, where you work through a list of options scribbled into your notebook. But the meat of the system is in the way you respond to what the NPC has to say. Therefore it is also the first game to explicitely separate asking and responding, illustrating the second area of improvement I will propose in chapter 4.

After questioning a suspect about a certain aspect of the case, you can either choose to believe their statement, doubt it, or if you have any evidence to the contrary, accuse them of lying. If you were right you unlock more clues, if you are wrong you stifle your progress and may end up sending the wrong man to jail.

**Parser-driven dialogue**

Back before games had visuals, and even a little while after, there was another type of dialogue: the text parser. Players would have to type in everything they wanted to say, and the dialogue system would do its best to translate that into one of the options that were predefined by the developer.

It sounds wonderful in theory, complete freedom of conversation, but in reality even to this day there is not a text parser that can perfectly understand and indeed parse every word in the human language. And even if it could, it would take developers years to fill up a database of matching responses. So they often compromise, directing similar keywords into the same response, or plainly ignoring unrecognized input. Naturally this sometimes results in misinterpretations from the parser, leading to frustration from the player – one of the reasons this system is rarely used anymore.

The last few parser usages of note have been ***Façade*** and ***Trilby's Notes***.

*Façade* (2005) is an experiment in narrative storytelling, driven by a very advanced text parser that moves beyond simple branching dialog to create a compelling emotional drama, centering around Trip and Grace, a couple that appear happy on the surface but really aren't, as the player soon discovers.


It works extremely well, giving you a compelling experience that feels real and free of restrictions, different on every playthrough. However the advanced technology is also its greatest weakness; on the rare occasion the parser does slip up, it immediately breaks the immersion. You can avoid this largely by 'playing along', sticking to the story and not trying to radically change the subject. But again this illustrates the biggest hurdle for parser-driven dialogues.

*Trilby's Notes* (2010) is an indie adventure game made by **Yathzee** (of *Zero Punctuation*[3] fame). It follows a retired thief on his investigation of a haunted hotel. Outfitted with a parser hardly as advanced as *Façade*, it still gets by surprisingly well.


I think the secret to the effectiveness of *Trilby*'s parser is that it is focused on simple, logical actions that are initiated through a handful of sanctioned keywords, mostly verbs such as LOOK, OPEN or ASK. This prepares players for the fact that these verbs are guaranteed to work while any others might not, averting some of the inherit frustration with parser-driven games.

**Systemic interactions**

This category is somewhat of a special case. What Ellison classifies as systemic interactions (2008) are dialogue systems that rely on simple actions to communicate with other characters. One notable example is ***Fable*** (2004), a fantasy RPG. The game has no traditional dialogue system of any kind, yet you can still communicate with NPC's via a host of *emotes*.

Boasting around rivals may make them dislike you, flirting with fair maidens wins their affection and eventually allows you to marry them if you wish. Any feedback comes from the body language of the characters. So without words or complicated menus the player is still able to interact with the people around him. And that is the general idea behind the systemic interactions classification.

[3] *Zero Punctuation* is a webshow where Yathzee reviews videogames in high tempo – hence the name.


***The Sims*** (2000) harbors a similar system, though it is a bit of a special case. While like *Fable* the game has no actual dialogue to speak of, it most certainly has a full-on dialogue system. It just does not seem like one because all it produces is gibberish.

Interactions with other characters in *The Sims* are actually somewhat similar to the *Hub-and-spokes* model. When you talk to an NPC a menu pops up containing a number of topics aswell as emotes.


You could talk to someone about art, mock their clothes or do a magic trick, and depending on the state of your friendship with this character they will react in a certain way, visualized by simple icons and expressive body language; not all that different from *Fable*. This system also shares a number of the same principles that found their way into my own dialogue system - but more on that later.

## 4. WHERE CAN WE IMPROVE?

Now that we have taken a look at how other developers have handled dialogue systems up to this point, we can start to formulate several areas of potential improvement.

### Freedom of movement

Many games still consider dialogue as a separate gameplay state. When dialogue begins, the normal gameplay rules no longer apply, and you are often locked in place for the duration of the conversation. This ofcourse helps the designer in staging the conversation, moving characters and changing camera angles, but it also hampers the player's freedom.

One of the few games that frees up dialogue a little more is *Heavy Rain*. During a conversation you can often still walk around in the vicinity of the character you are speaking to, and there are a few special interactions available in the environment where you can lean against a wall or sit down on the couch while you talk. Even this limited control makes it feel more like you are an active participant in the conversation instead of just a spectator. It is a good step forward but I feel like more can be done to improve the player's freedom of movement, better integrating dialogues with the rest of the game.

### Two-way conversations

As mentioned earlier, dialogues are often strictly a one-way conversation. The player questions an NPC and they respond, rarely asking questions themselves or contributing in any way to the conversation. The illusion of two-way conversation comes from the player character's response, which oftentimes is completely scripted, offering little to no player input.

For dialogue to feel more natural, I think NPC's should play a more active role in the conversations, and there should be a seperation between asking things and replying to things, giving players the tools to manage both directions of a conversation. We can see examples of this in games like Mass Effect and LA Noire, but I think there is still plenty of room for improvement.

### Flow

Game conversations are often very stop-and-go. When the player is confronted with the list of options the conversation falls flat, idle until the player makes a choice (aside from exceptions like *Alpha Protocol*). Presenting the options during the dialogue is a good improvement, but players still need to parse the meaning of these options before they can decide, particularly if they are presented as keywords. This break with the *flow* of the game, a mental state of immersion in the current activity, where actions come natural and without thinking actively about them. (Wikipedia, 2011a)

To improve the flow of game conversations I think dialogues should be constructed in a more logical manner, closer to the way we think and construct sentences, and perhaps more goal-oriented. Giving the player a clear goal in a conversation can help them formulate a rough idea of what they would need to ask or say to get to that goal, making dialogue navigating easier and more intuitive.

## 5. UNDERSTANDING CONVERSATIONS TO BUILD BETTER DIALOGUES

Up until now we have been doing pretty well with our dialogue systems, and a large part of that can be credited to good writing. Because no matter how solid the technology, if the writer is no good the dialogue won't be either. But, as became evident in the previous chapter, even with good writing there is still something missing from current dialogue systems, some fundamental factor that makes the conversations feel more natural and believeable.

To find out what that missing factor is, I think we need to stop relying on what we think we know about conversations and turn to the field of **linguistics**.

Linguistics is defined as **the scientific study of human language** (Wikipedia, 2011b). It looks at how we structure our sentences, the different meanings our words can have depending on context, and the way language evolves over time. For the purpose of understanding conversations, I think structure (*syntax*) and to some extent meaning (*pragmatics*) will be the most interesting aspects.

Upon searching for any sort of scientific document that would help me understand these aspects of conversations better, I found a research paper called '*An introduction to Conversation Analysis*' (2007) written by professor **Anthony J. Liddicoat**, a professor of Applied Linguistics at the University of South Australia. In this paper Liddicoat relates and augments a number of models about the structure of a typical conversation, the most prominent one of which was drawn up by linguists Sacks, Schegloff and Jefferson in '*A simplest systematics for the organisation of turn-taking for conversation*' (1974) after they had carefully analyzed a large amount of real-world conversations between regular people. I will relate several key elements of this model that I found particularly useful when constructing my own dialogue system.

### Turn Constructional Units

Sacks et al. (1974) posit that all conversations are made up of *Turn Constructional Units* (TCU). A variety of grammatical units may function as TCUs: sentence, phrases or even a single word. An example from one of the many conversation transcripts in the paper:

> *Therapist: What kind of work do you do?*
> *Woman: Foodservice.*
> *Therapist: At?*
> *Woman: Uh the post office cafeteria on Redwood.*
> *Therapist: Okay.*

The word *At?* in this case functions as a complete TCU, meaning that this single word in and of itself functions as a full unit at this point in the conversation. Subsequently it is recognized as a sufficient unit by the woman, who produces a response.

TCU completion is measured in three ways: Firstly, the unit may be *gramatically* complete, such as a full punctuated sentence. Secondly, it may be *intonationally* complete, meaning the intonation indicates the end of the TCU, such as when someone is being stared at and they ask "What?!" in an annoyed tone. Thirdly, and most importantly, the TCU has to be complete as an *action*: it must have done what it needs to do at that point in the conversation. For example, an answer usually follows a

question. However it may also be that a person avoids the question by changing the subject, something that can be recognized by the other person as avoidance, another acceptable action at that point in the conversation, thus completing the TCU.

This explains why people sometimes attempt to finish eachother's sentences; most people have an intuitive understanding of these conversational units, allowing them to make predictions on their possible completion. They expect a certain outcome based on context and previous experiences.

This intuitive understanding of where a TCU begins and ends becomes important when looking at turn-taking in conversations.

**Transition Relevance Places**

*Transition Relevance Places* (TRP) are the spaces inbetween TCUs where it becomes acceptable for a person other than the current speaker to take over the speaking turn. This does not mean that the other person WILL assume the speaking turn, or even that the current speaker is done talking, it merely suggests that IF it were to happen, it would be acceptable judging by the sentence structure.

Sometimes it is not immediately clear who the next speaker will be, as illustrated by this example:

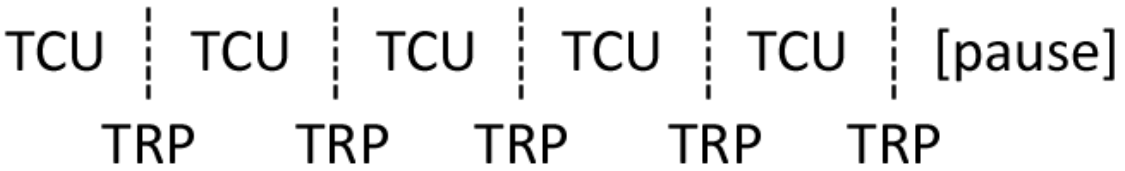
*Man: Can you pass the butter?*

With several people in the room, the man's question is not inheritly directed at a specific person, but rather relies on environmental cues such as the location of the butter in relation to the other participants. Whoever is closest should perform the required next action. This person will then take **ownership** of the conversation to produce the next TCU.

Liddicoat also presents a set of rules Sack et al. produced using this theory (2007, p68), which almost reads as a code structure for a dialogue system:

1. At any Transition Relevance Place following an initial Turn Constructional Unit:
   a. if the current speaker selects the next speaker, then the selected person has the right and obligation to take the next turn to speak, no other speaker has such rights or obligations;
   b. if the turn so far is not constructed to select a next speaker, then self-selection may, but need not occur. The first participant to begin speaking acquires the right to a turn;
   c. if the turn so far is not constructed to select a next speaker, the current speaker may, but need not continue if no other speaker self-selects.
2. If the current speaker continues after the initial TCU, these rules apply again at the next Transition Relevance Place aswell as each subsequent one until speaker change occurs.

In this model, each person is allowed to complete one TCU before the first TRP presents itself.



Here person A produces six TCUs before pausing to let person B take the speaking turn. Person B may also legitimately assume the speaking turn at each TRP.

However it could also be that person B attempts to claim the speaking turn *inbetween* TRPs, cutting off the current speaker. When this happens, either person A will let this happen or the conversation will become competitive as the two attempt to work out who is to become the next speaker. You have probably experienced this yourself: both speakers continue speaking in short bursts until one of them gives up and relinquishes the speaking turn. At this point we stumble across a third elements: *Rhythm*.

**Rhythm**

When two speakers overlap, they begin to speak according to a certain rhythm. This can roughly be linked to the syllables in each word, where every syllable represents a '*beat*' in the sentence and the conversation - something that becomes very apparent in rap music for instance. Liddicoat actually uses the word 'beat' to describe this phenomenon (2007, p.96), as does a recent article from Kotaku called '*The Rhythm of Play*' (2011), which compares flow in games to rhythm in music, reaching the same conclusion I did while researching this phenomenon, namely that the two are unequivocally linked.

The appropriate word for this is **cadence**, a term found both in linguistics and music, referring to a *melodic or harmonic pattern that indicates the end of a sentence or piece of music* (Wikitionary, 2011). This goes back to the idea of TCUs and how one can predict their possible ending ahead of time - not unlike the way *Mass Effect* presents possible replies before the end of the last sentence.

When you learn to recognize the cadence in a conversation, you start to see the emergence of *flow*.

**Flow**

In a blog post on his website (2009), **Krystian Majewski**, game designer from Germany, tells us that he believes multi-choice dialogues don't work. Their gameplay is so different from what we consider to be regular gameplay that it breaks the flow of a game.

> "Basically, multiple choice dialogue will process only very little information from players and won't give them clear feedback. It won't create a mutual information exchange, won't create a flow state and won't involve the players emotionally."
>
> *- Majewski 2009*

He uses *Emerald City Confidential* (2010) as an example, an indie adventure game marketed towards casual gamers. During playtesting the developers discovered that their branching dialogue system confused the hell out of players. The casual gamers were used to there being only one right solution to a given situation and the rest would mean *game over*. They applied the same logic to the dialogue, where there is no right or wrong, and were subsequently terrified to choose an option because they had no idea if it was the right one.

Here Majewski unearths another problem with contemporary dialogue systems – gamers are used to performing actions over and over again. It is the basis of most regular gameplay as we know it.

Run, take, cover, shoot, jump, run, take cover, shoot, reload, run, shoot... players perform these actions so often it almost becomes second nature to them. No longer do they have to think about which button to press or how far to aim next to an enemy to compensate for recoil, they just… do it.

And that is where *flow* comes from. *Repeatable actions.* Recognizeable patterns. (again: cadence)

**Repeatable actions**

Most of our gameplay is based around this principle of repeatable actions. We see it in action games, racing games, and even in the Sims' dialogue system. Repeatable actions give us something to learn, something to get better at, faster, smarter. And once we've mastered them, they create flow – one of the reasons why games with repeatable actions are so addicting. (Chou & Ting, 2003)

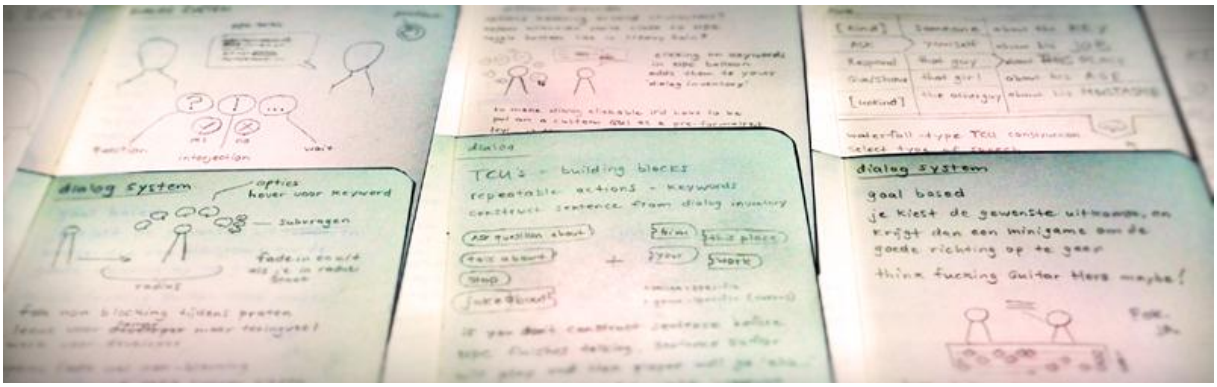
Most dialogue systems, on the other hand, present us with *unique* actions every time.

You might be able to predict roughly what the outcome of a certain option will be in for instance *Dragon Age II*, where the middle option always triggers a joke. But even then NPC's might respond differently based on their personalities or how much they like you. Add to that the pervading trend of shortening dialogue options into keywords, and players end up having no idea what they are about to say when they pick a dialogue option.

It became clear to me that the best way to make dialogues flow more naturally was to employ the idea of repeatable actions. This became the cornerstone of *ConFlo*, the custom dialogue system I designed.

## 6. THE ROAD TO CONFLO IS PAVED WITH PROTOTYPES

In order to find out if my thoughts and theories regarding a better dialogue system would work in practice, I developed several prototypes to put them to the test.



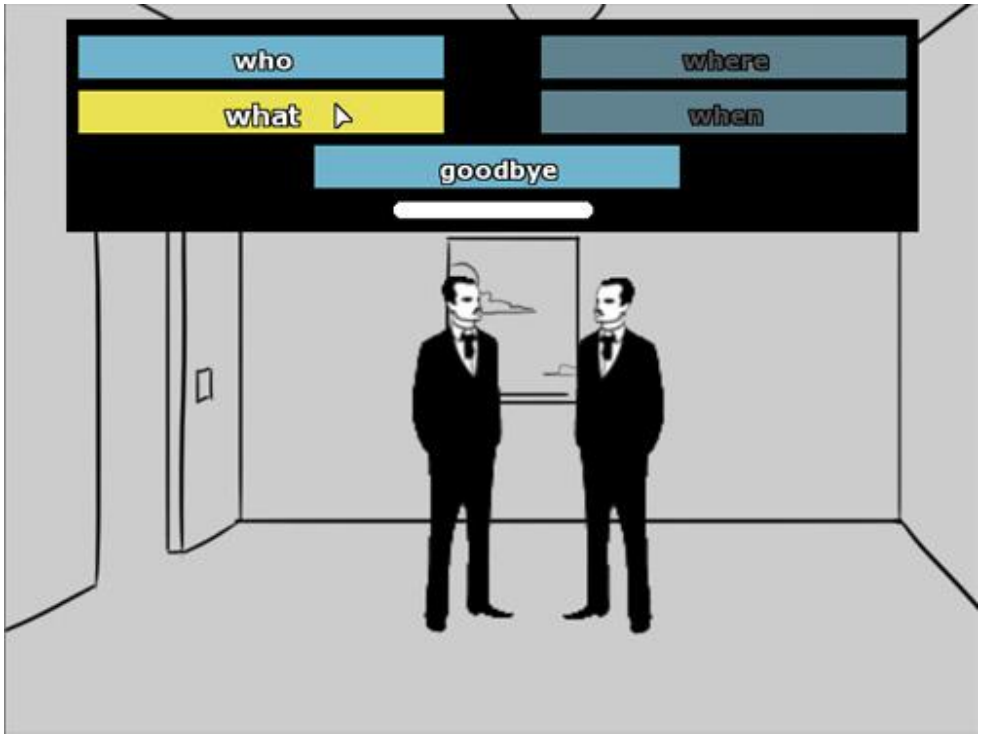

In order to build these prototypes I used **Adventure Game Studio**, a piece of software with a decade of development already under its belt, designed specifically to create adventure games. I have used AGS extensively in the past, so my experience with this engine coupled with the fact that it is simply the best engine for creating the genre of games I focus on in this thesis made AGS the perfect candidate for a prototyping environment.

**Prototype I**

I started out by emulating the latest developments in dialogue systems in AGS. Since I had never built a completely custom dialogue system before, this would help me understand the structure involved in building one.

I chose to borrow primarily from *Alpha Protocol*, as it already mixes together elements from games like *Mass Effect* and *Indigo Prophecy*. The specific features I recreated were the **limited options** (I chose to use four buttons, excluding a 'goodbye' button), the **stance-based responses** (professional, suave, aggressive etc), and the **time limit**. The time limit especially I had never seen being used in adventure games, so I was curious how well that particular feature would work. In addition to that I wanted to experiment with letting the player **roam freely during conversations** instead of being locked in place for the duration.

Because I knew this was going to involve a lot of advanced coding, I decided to keep a step-by-step todo list *slash* changelog. This proved to be very useful in helping me divide this humongous task into manageable bits, and keep track of which custom functions I used to build them. An excerpt of it can be found as appendix of this thesis.




The initial prototype came together pretty quickly, thanks to some plugins that were developed by the AGS community. With the *Countdown* plugin I was able to easily set up a timer and link it to a resizeable bar to achieve the *Alpha Protocol* style time limit. I also built an auto-select system so there would always be an option selected when the timer ran out. But where things got really interesting was the *QueuedSpeech* plugin.

AGS is based on a modified version of C++, and the lion's share of code in AGS is **blocking**. *Blocking* means that the command in question blocks further execution of the script until it has finished doing its thing. *Walk* is a blocking command for instance, so if I say *Walk(x, y)*, the script will

wait until the character reaches those coordinates before it continues. Sometimes that is desirable, but sometimes it is also nice to be able to do things in the meantime.

In *Mass Effect* for instance you can do nothing else but stand stiffly across from eachother during a conversation, so I wanted to try making the dialogue system **non-blocking**, meaning you can walk around and even interact with objects during a conversation. And for this I needed *QueuedSpeech*, because while AGS has a BackgroundSpeech function (a non-blocking way of saying things), it can only handle one line at a time, which is no good for conversations. *QueuedSpeech* fixes that, plus a queue structure was great to keep track of the conversation if the player interrupts the other person (which could happen quite a lot if he is allowed to freely interact with the world during the dialogue). And so it became the backbone of this first prototype.

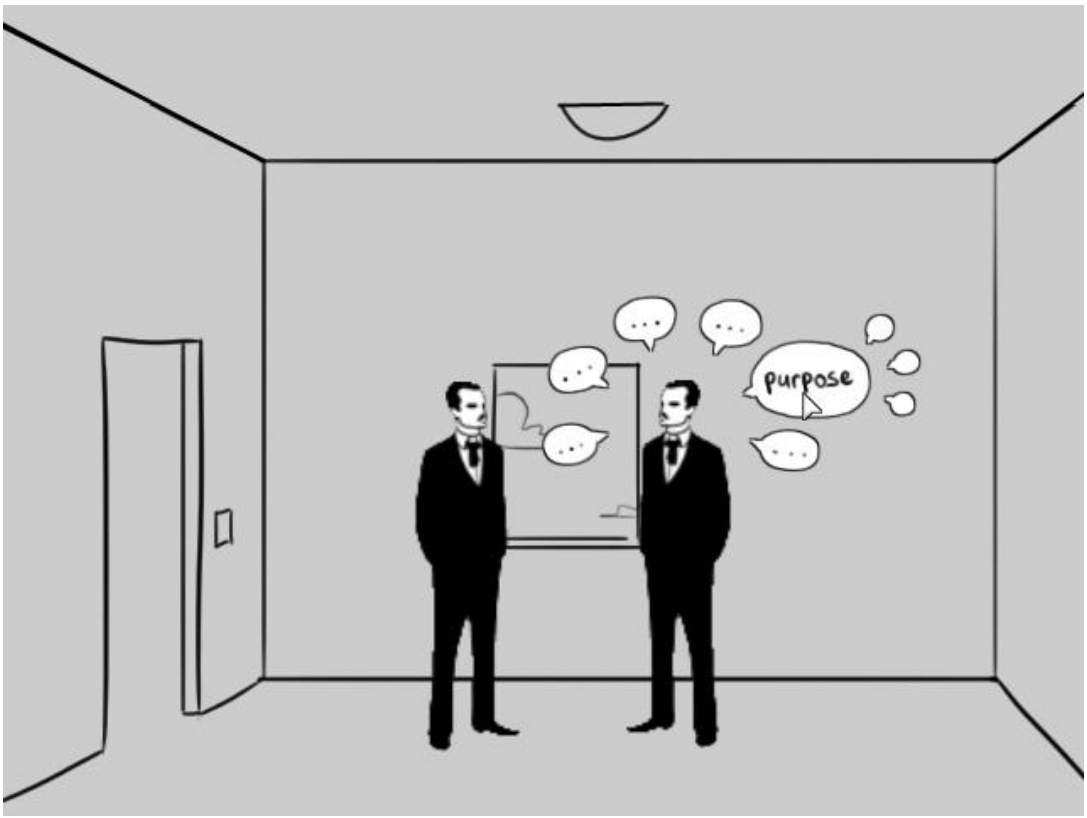

20 steps, 32 substeps and a sizeable amount of changelog pages later, I had the system up and running. To fieldtest it, I decided to use it in a short game (*15 Minutes*, 2011). Immediately this revealed a slew of problems I had not accounted for in the testing environment, consisting mostly of clashes between the blocking and non-blocking code. I saw no solution to them (that was possible in AGS), and coupled with the fact that it was hard to create a good narrative when the player could be messing around on the other side of the room at any time made the viability of this feature questionable. It occurred to me that while non-blocking dialogues were a nice idea in terms of immersion and freedom, they take away from the main goal of most dialogues in adventure games, which is **conveying information and furthering the story**. So I decided to retire that feature.

Some players also complained that the timer was too fast, so I extended it. Other than that the system seemed to be working well and among the dozen or so testers it was generally found to be intuitive and refreshing.

**Prototype II**

So now I had a system that worked, but it was hardly pushing the boundaries. Though not widely adopted yet, most of these features existed seperately in other games, and even grouping them together they were still essentially a list of options to work your way through. So I started conceptualizing other kinds of gameplay for dialogues.

Prototype 2 looked at a different way of presenting the options and their cohesion. Taking a note from *The Sims*, I presented the options as balloons containing topics, with smaller suboptions grouped around each one. This really takes advantage of the idea of **repeatable actions**, placing topics in the same spot each time and having a predictable set of suboptions attached to them that can all be navigated quickly via a hub-and-spokes like visual menu.




While visually pleasing, the main issue I had with this system was that it felt very limiting. You could have maybe five or six balloons around a character at one time, all consisting of only one keyword and about three suboptions, otherwise things got too crowded. Plus it was not *really* that much different from the standard list of topics yet.

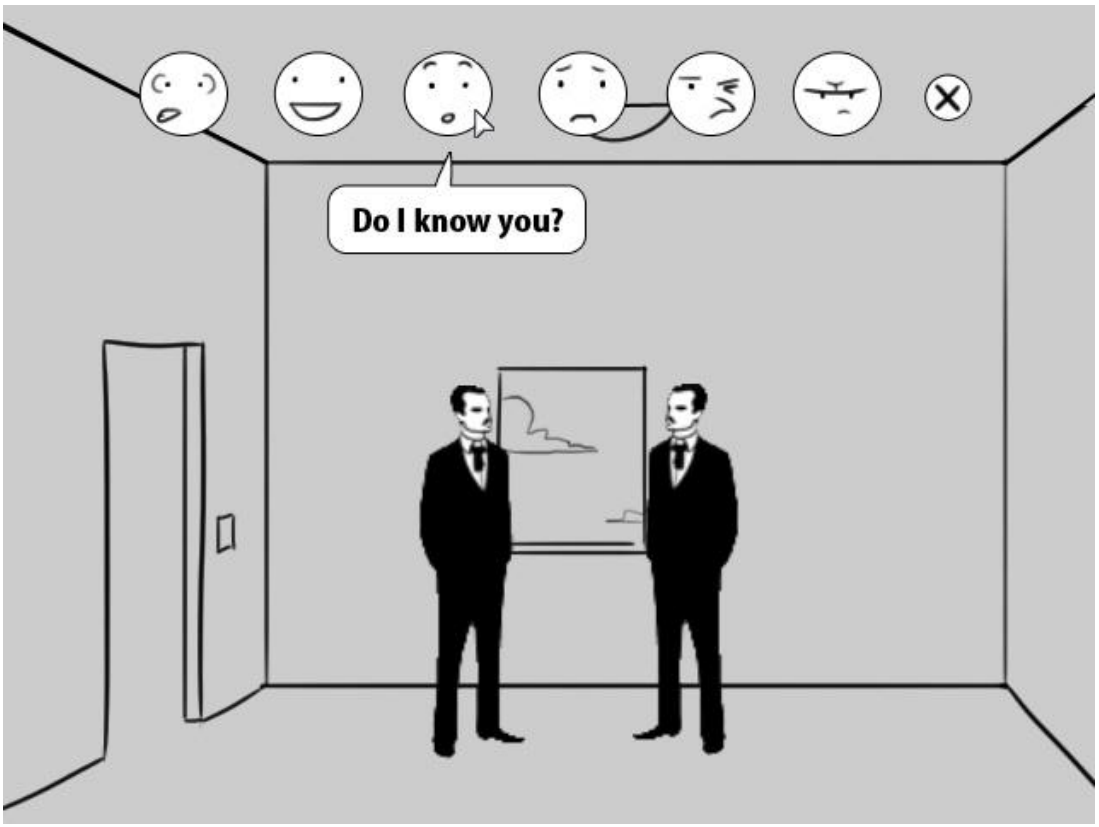
**Prototype III**

The next prototype put more focus on choosing *how* you say something instead of *what* you say. I wanted to move away from keywords for a second because a common complaint with them is that they offer no real insight into what a character is going to say, and more importantly with which tone. What the player has in mind when he clicks a keyword may be radically different from what the

designer had in mind, leading to a break in immersion. So I decided to turn things around and focus on *emotions* with this prototype.

In the *Facial Action Coding System* (1978), psychologist **Paul Ekman** describes a list of what he believes are the six basic human emotions: sadness, happiness, anger, fear, disgust and surprise. Ekman developed this theory by studying the facial expressions of the Fore tribesmen in Papua New Guinea. He observed that members of this isolated culture could reliably identify the expressions of emotion in photographs of people from cultures with which they were not yet familiar. From this evidence, Ekman concluded that the expressions associated with some emotions were basic or biologically universal to all humans (Ekman & Friesen, 1969, p.49-89).

In this prototype I represented these emotions with a set of abstracted facial expressions (smileys). I also turned *surprise* into c*uriosity* to create a default emotion for asking questions.




During a conversation the player can choose the emotion that matches his own feelings at that time. Hovering over the smiley would display the full line once players make their selection, to make sure the emotion they chose matches the actual thing they wanted to say. This method is also found in *Desperate Housewives The Game* and to some extent in *Deus Ex Human Revolution*.

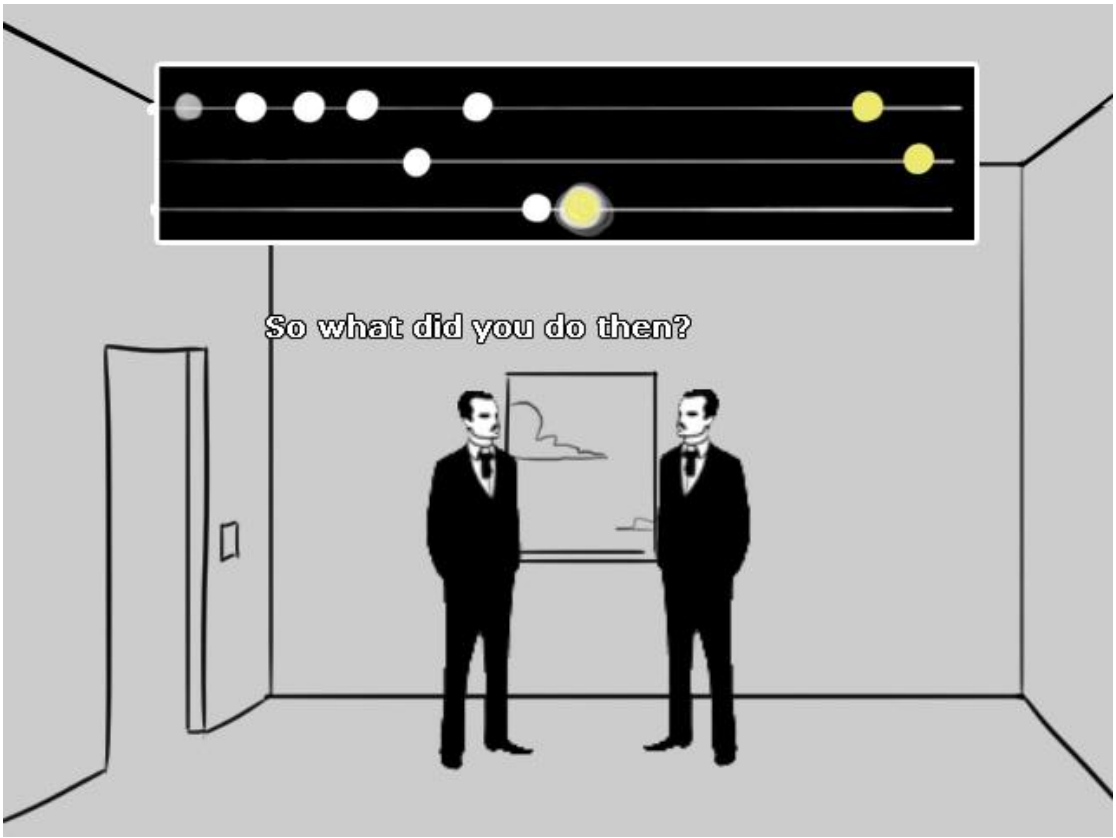
Filling the six categories equally proved to be a challenge. Some were overused, such as *curiosity*, and some were rarely used at all, such as *disgust*. Plus sometimes a response did not fit in any category. I concluded that responses based on emotions and stances were only useful if they were specifically tailored to the topic at hand, and even then they would merely be visual aids to suss out the tone of each response with little added gameplay value.

**Prototype IV**

Prototype 4 was something radically different, based on my aforementioned theory of flow and rhythm. I wanted to incorporate patterns and music into it, so I took the *Guitar Hero* control scheme as a starting point and tried to reshape it as a dialogue system.

The initial idea revolved around a few 'tracks', which might correspond once again with a number of emotions or stances, between which the player could then switch during the conversation by using a button combination. And maybe a series of dots would come down each track to represent an obstacle in the conversation or a place where the player should interject. Maybe the patterns for one stance were easier than the other, but would have a bigger payoff.

There were a lot of maybe's.



I liked the idea of it but in the end my lack of musical knowledge quickly stifled me, as I could not come up with a combination that felt good and natural, much less put it into practice with my limited programming skills. I also feared a minigame would become too much of a distraction to the player. I would not want them to focus all their attention on the patterns and then completely miss the dialogue. It felt too much like a gimmick at that point so I abandoned it.

**Prototype V**

With prototype 5 I went back completely in the opposite direction.

Creating a more natural way to converse with virtual characters was all well and good but I had to consider my target market: adventure games. And the focus of adventure games is still the story. These prior experiments were all interesting but I felt they impaired my ability to effectively convey a narrative, taking attention away from the *what* in favor or the *how*. So I decided to try a different angle.

Working off of Liddicoat's theory of conversational construction units I wanted to see if it was possible to build a system in which you could construct your own TCUs. To a degree ofcourse, I could hardly make sections for vowels, adjectives, nouns and the like, the possibilities would be endless. So I set out to find a handful of common categories from which you could construct an array of different conversations.

The **Ask** category was a no-brainer, but I knew I also wanted to get a separate **Respond** category in there. As I mentioned before, most adventure games still revolve mainly around asking questions, but rarely do you get to truly respond to things the other characters say. Yet those two cover most of the common occurences in conversations – you either ask something or you respond to the other person. Following that logic there was only one category missing: **Tell**. Because sometimes you say things that are not in direct reply to something else.

Between that and the other two categories it looked like I had my bases covered. I added a fourth and final category to this list, called **Give**, which would bring the items you are carrying directly into the conversations, and you could exchange or discuss them there.

The inspiration for the interaction design of this prototype came from an unusual source: ***CounterStrike*** (1999). CS is an First-Person shooter made by *Valve Software* pitting a team of terrorists against a team of counter-terrorist soldiers. My inspiration came from the ingame store:


At the start of each round, players are presented with a store interface in which to buy weapons and equipment. This store is built up of a few different categories that open up into submenus. Novice players will open up each category, look at the stats on all the items, then make their decision, while experienced players often have a preferred loadout and know exactly where to find those items.

Additionally, this store interface has keyboard shortcuts, so once experienced players memorize the location of their preferred items and the matching keyboard commands, it takes them only a second to buy everything they need. 1-4, 4-3, 6, 6, 6, 7, 7, 8-2, 8-5, done. Those numbers may mean nothing to you but an experienced CS player can tell you exactly what that keycombo just bought them.

I realized the reason the CS buy menu works so well is because it also uses the principle of *repeatable actions* as described in chapter 5. Experienced players have memorized the location of each item and thus can reach them much quicker than someone who just encountered this menu for the first time. I wanted to apply the same principle to my dialogue GUI[4], so I created this:

[4] Graphical User Interface


The GUI is formatted in a way that accomodates the most common actions player will want to perform during a dialogue – do you want to ask a question? Or tell somebody something? Looking at the GUI you will know exactly where to go.

Once you make that choice you have to decide who you want to adress. Up to now most of these prototypes were based on character-to-character interactions, but this time I wanted to detach dialogues from being purely single-person-centric, bringing back a little bit of that non-blocking-ness I had discarded after the first prototype. I decided to make it so that you could pull down the dialogue GUI at any time, anywhere. You don't even have to be close to another person. And in addition to being able to adress anyone currently in the room with you, you can also adress **all of them at once**, or even adress **yourself**.

They say talking to yourself is a sign of madness, but I saw some interesting gameplay usages for it. First of all, it could be used as **hint system** of sorts. The player asks himself why he is doing what he is doing, which can subtly remind them of their current objective. It could also serve as a gameplay mechanic, where for instance the main character is a shy person and has to talk courage into himself whenever he is faced with an obstacle to overcome. There are plenty of interesting examples to come up with there.

The last column is then filled up with the topics you can discuss. And under the Respond tab you can find what are essentially a bunch of emotes, such as Laugh or Insult, as well as Agree or Disagree, standard replies which can be used whenever another character asks for your opinion on something.

And once you start a conversation, you should keep it going, because the people you talk to do not possess infinite patience. They will get annoyed if you fall silent during a conversation, and might

eventually walk away if you do not respond anymore (unless ofcourse you explicitly bid them farewell by closing the dialogue GUI).

Together this forms a kind of **waterfall system** through which players can quickly construct different kinds of replies. Recurring options are always placed at the same location, so once the player becomes familiar with the system they should be able to quickly formulate exactly the kind of reply they are looking for.

Because of this waterfall-like structure, I decided to name the system **ConFlo**.

## 7. PLAYTEST RESULTS: HOW PEOPLE FEEL ABOUT CONFLO

To put my system to the test, I used it in a few different prototypes - or *situations* if you will. In each situation the player is presented with an objective, and they have to reach that objective by communicating with one or several characters present in the room. I chose to make several situations because I did not only want to see if players would understand the ConFlo GUI, I also wanted to see if they would become more adept at navigating it the more time they spent with it.

*The first situation* consists of a guy seeing an attractive girl at the bus stop. He would like to ask her out but at the onset he is too shy to do so. You can overcome this using the dialogue system, and then fire off a few questions to get to know the girl. This alone will not get you there however, and you will have to use the Respond tab well in order to win her over. Otherwise you will run out of things to say and miss your shot.

*The second situation* takes place at a crime scene, where a detective duo looks around for clues, exchanges theories and questions suspects. While the previous prototype stayed closer to classic adventure game conversations (a lot of asking questions), this situation allowed me to test drive every aspect of the ConFlo system. Items the player collects become available in the GUI, and you can show them to your partner in order to connect the dots and deduct clues. These clues then show up in the list of things to ask the suspects. You could even decide to play good cop-bad cop, where one of you asks the questions and the other responds positively or negatively to the answers the suspects give. Then afterwards you can confer with your partner and tell him who you think did it.

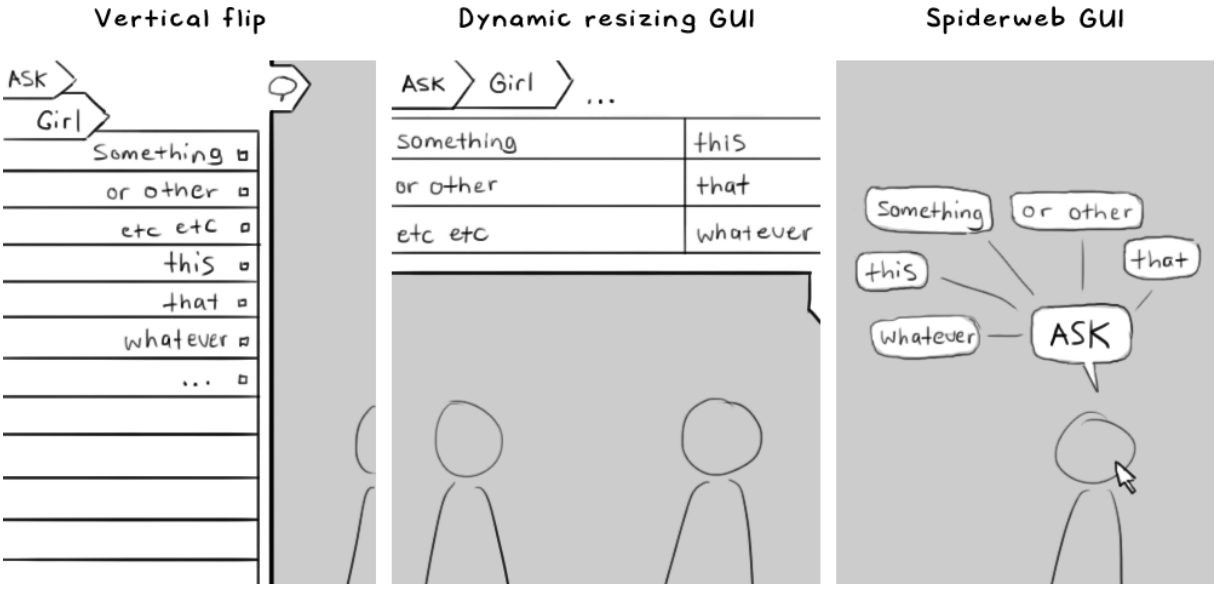
*The third situation* depicts a boyscout trying to convince his friends to help him with something. It will be part of an adventure game called **Trusted Soil** that I have been developing alongside ConFlo these past months with a classmate. It marks ConFlo's first integration with a full-fledged game.

### Improvements made

Following these playtests and my own continued use of the system I amassed a nice list of points to improve upon, both from the developer side and the player side.

First and foremost, while the GUI looked and performed great, it was a tad oversized. This could become a problem if characters are near the top of the screen, because their speech text would appear behind the dialogue GUI - something I had overlooked when I first created it in Photoshop. So

I tightened things up as much as possible and in addition conceptualized a few different designs for the GUI that were more space-efficient and/or vertically oriented.




I also found the *EVERYONE* category very hard to use effectively, plus I had no place to store it in the array since the range it belonged to was dedicated exclusively to character ID numbers; short of creating a separate invisible character just for that purpose there was no way to fit it in, so I cut it.

Furthermore, in writing the dialogue, I came across some situations where a *YES/NO* response would be preferred over an *AGREE/DISAGREE* response, such as when being asked if you know someone. I considered rephrasing them into *AFFIRMATIVE/NEGATIVE*, but that pairing didn't sound right, so I took it to a more abstract level by naming them *THUMBS UP/THUMBS DOWN*.

And aside from minor fixes related to the prototypes themselves, there were also a few options in the Response tab that felt out of place. Most of the options follow a logical sentence construction, such as *ASK > GIRL > FOR HER NAME*. After the first prototype I found that several options in the Response tab broke with that logic. C*OMPLIMENT*, for instance, was under *RESPOND > POSITIVE*, while according to logic, it should be under *GIVE*. Similarly with *JOKE*; you could use it to make a quip in response to something another character said, but used standalone it would fit better under *TELL*.

So I restructured things a bit. C*OMPLIMENT* and *JOKE* seemed like incidental things rather than being useful all the time, so I left those out for developers to add manually to *GIVE* or *TELL,* respectively. With the *POLITE* and *BLUNT* categories now all but empty, I merged them with *THUMBS UP and THUMBS DOWN*, renaming them into *POSITIVE* and *NEGATIVE*. Then I filled the void *JOKE* left with *PLAYFUL*. While this simplification seems to take away a bit of the diversity in the Response tab, it actually increases it by freeing the responses from hard keywords such as LAUGH or INSULT. Positive or negative are broad terms, so they can dramatically increase the number of possible nuances in the responses, while also decreasing the number of hard options that the developers need to fill up.

These adjustments made the system a little easier to use and understand, though they only just begun to scratch the surface of what would be possible with more time and thought.

## 8. WHAT WE CAN LEARN FROM ALL THIS

Looking back on all prototypes and comparing them to my research question reveals that there are two prototypes that come closest to what I set out to achieve - namely a system that feels natural and better simulates the nuances of real conversations. And those are prototype I and prototype V.

### Prototype I

Pros:
- easy to understand
- runs by itself even with minimal input
- simple to control

Cons:
- very limited number of options
- fully blocking due to technical limitations in AGS
- confusing code structure and generally a lot of work to maintain for developer
- not really unique or innovative

The first prototype was also one of the best ones. It was easy to understand and simple to control, and would move forward even with minimal input, something players aswell as myself greatly enjoyed. Unfortunately behind the scenes it was rather messy and required a lot of work to maintain. This could be improved by rewriting the system in a more capable engine, but because my focus was Adventure Game Studio and I could not make it work there, it was not my preferred prototype.

In terms of innovation it was also hardly original, as it was more or less a direct copy of the *Alpha Protocol* and *Mass Effect* systems. Though it was mainly intented as a learning experience for myself, it proved that this system works very well, improving on the default branching system while not deviating from it completely.

### Prototype V

Pros:
- relative freedom of movement
- more accurate representation of the most common nuances of real conversations
- access to every dialog in the room at any time and any place
- relatively easy to maintain code structure
- developer effort is scalable, accomodating both many unique options aswell as lots of re-use

Cons:
- players may have trouble keeping track of conversation state when switching often between tabs
- not all aspects of the system will be useful for every game
- a little more work for the developer to optimally exploit all aspects of the system

Even within the span of a single prototype I could see players were getting the hang of the waterfall structure fairly quickly. Even if they forgot exactly under which tab a certain option was, they did remember that it existed, and went looking for it. Ironically it seemed the inexperienced adventure

game players were quicker to grasp the structure and flow of the GUI than the hardcore players, presumably because the hardcore players are so used to seeing the default branching system.

While it may not be the most ideal system for every game, it is the one that best fits my original research question. It still has a lot of room for improvement so I do not consider this its final incarnation, and will continue to improve on it in the future.

If you would like to try out prototype V for yourself, it is available for download from http://timhengeveld.com/files/conflo_beta.zip

In general, I think we as an industry are still a ways off from fully realizing the idea that drove me to write this paper and do this project, namely to create fluid and realistic conversations in videogames. Conversations are just so incredibly diverse and intricate, they cannot easily be replicated virtually without at least a great writer and a lot of work. There are so many factors – in this thesis we took a look at how players can control the dialogue, but we haven't even touched on things like body language, intonation, context, meaning, pre-existing relationships between speakers; the list goes on.

But still I hope to have made a small contribution to the future of dialogue systems with this project, if not with my prototypes then at least with the theories behind it. Hopefully more developers will come to realize that linguistics is a useful and powerful companion in the evolutionary process of the dialogue engine. Once we embrace all that linguistics has to offer us can we really move forward with interactive fiction.

So I think it is safe to say that when it comes to dialogue systems the last word has not been spoken.

Thank you for reading.

**BIBLIOGRAPHY**

*Alpha Protocol*. 2010. [PC, Xbox 360, PS3]: Obsidian Entertainment.

*Broken Sword: Shadow of the Templars*. 1996. [PC]: Revolution Software.

Bronstring, M., 2003. *Feature: The Future of Adventure Games*. [Online] Available at: http://www.adventuregamers.com/article/id,318/p,6 [Accessed 21 May 2011].

Bronstring, M., 2007. *Blog: Why adventure games should steal Mass Effect's dialog system*. [Online] Available at: http://www.adventuregamers.com/blogitem.php?id=22 [Accessed 21 May 2011].

Chou, T.-J. & Ting, C.-C., 2003. *The Role of Flow Experience in Cyber-Game Addiction*. Adelaide, Australia: School of Marketing, University of South Australia.

*Counter-Strike*. 1999. [PC]: Valve Software.

Cruz, U.S., 2011. *Prom Week*. [Online] Available at: http://games.soe.ucsc.edu/project/prom-week [Accessed 21 May 2011].

Delaware, D., 2010. *Question on the dialogue system*. [Online] Available at: http://social.bioware.com/forum/1/topic/141/index/3086433/1 [Accessed 21 May 2011].

*Dragon Age*. 2009. [PC, Xbox 360, PS3]: Bioware Corp.

Dragt, I., 2009. *Game Dialogue Systems*. Hilversum: Utrecht School of the Arts.

Ekman, P. & Friesen, W., 1969. *The repertoire of nonverbal behavior: Categories, origins, usage, and coding.* Semiotica.

Ekman, P. & Friesen, W., 1978. *Facial Action Coding System: A Technique for the Measurement of Facial Movement.* Palo Alto: Consulting Psychologists Press.

Ellison, B., 2008. *Defining Dialogue Systems*. [Online] Available at: http://www.gamasutra.com/view/feature/3719/defining_dialogue_systems.php [Accessed 21 May 2011].

*Emerald City Confidential*. 2010. [PC]: Wadjet Eye Games.

*Fable*. 2004. [PC, Xbox]: Lionhead Studios.

*Façade*. 2005. [PC]: Michael Mateas & Andrew Stern.

*Guitar Hero*. 2005. [PS2, Xbox 360, PS3]: Harmonix.

Hamilton, K., 2011. *The Rhythm Of Play*. [Online] Available at: http://kotaku.com/5808033/the-unsung-musical-secret-of-great-gamesand-how-some-games-get-it-so-wrong [Accessed 3 June 2011].

*Heavy Rain*. 2010. [PS3]: Quantic Dream.

IRIS, 2011. *Interactive Storytelling and Narrative Theories*. [Online] Available at: http://tecfalabs.unige.ch/mediawiki-narrative/index.php/Interactive_Storytelling_and_Narrative_Theories [Accessed 21 May 2011].

Juuso, 2010. *Goal-based dialogue system*. [Online] Available at: http://www.gameproducer.net/2010/11/27/goal-based-dialogue-system/ [Accessed 21 May 2011].

*LA Noire*. 2011. [PS3]: Rockstar/Team Bondi.

Liddicoat, A.J., 2007. *An Introduction to Conversation Analysis*. London: Continuum.

Majewski, K., 2009. *Reinventing Dialogue*. [Online] Available at: http://gamedesignreviews.com/scrapbook/reinventing-dialogue/ [Accessed 21 May 2011].

*Mass Effect*. 2007. [PC, Xbox 360, PS3]: Bioware Corp.

Mateas, M. & Stern, A., 2003. *Facade: An Experiment in Building Fully-Realized Interactive Drama*. [Online] Available at: http://www.interactivestory.net/papers/MateasSternGDC03.pdf [Accessed 21 May 2011].

Moeller, J., 2010. *Daniel Erickson Discusses BioWare Dialogue Systems*. [Online] Available at: http://www.starwarsmmo.net/news/daniel-erickson-discusses-bioware-dialogue-systems/ [Accessed 21 May 2011].

Sacks, H., Schegloff, E.A. & Jefferson, G., 1974. A simplest systematics for the organisation of turn-taking in conversation. *Language*, 50, pp.696-735.

Short, E., 2009. *Analysis: Conversation Design in Games*. [Online] Available at: http://www.gamasutra.com/php-bin/news_index.php?story=23559 [Accessed 21 May 2011].

Sylvester, T., 2008. *Designing 'The Player League' Part 2: The Mechanics*. [Online] Available at: http://tynansylvester.com/2008/designing-the-player-league-part-2-possible-mechanics/ [Accessed 20 July 2011].

*The Secret Of Monkey Island*. 1990. [PC, Mac, Amiga]: Lucasarts.

*The Sims*. 2000. [PC]: Maxis.

*Trilby's Notes*. 2006. [PC]: Yathzee.

Wikipedia, 2011a. *Flow*. [Online] Available at: http://en.wikipedia.org/wiki/Flow_(psychology) [Accessed 26 July 2011].

Wikipedia, 2011b. *Linguistics*. [Online] Available at: http://en.wikipedia.org/wiki/Linguistics [Accessed 26 July 2011].

Wikitionary, 2011. *Cadence*. [Online] Available at: http://en.wiktionary.org/wiki/cadence [Accessed 26 July 2011].

**APPENDIX**

# ConFlo - conversational flow engine
## Technical document

SCUMM-like menu that allows you to construct questions and responses.
Accessible at any time by the balloon icon.

Contains a few categories with sub-categories and options, divided in:

**ASK** > select WHO to ask >
- game-related queries (locations and objects; where is the bathroom? who has the key?)
- person-related queries (where have you been? how was your day?)

**RESPOND**
- agree
- disagree
- unsure?
- polite > compliment/flirt/joke/laugh
- impolite > insult/punch/threaten

**TELL** (?) > select WHO to tell
- explain
- tell a story

**SHOW/GIVE**
- exchange items

**name suggestions:**
DDS - Dynamic Dialog System
DCS - Dynamic Conversation System
FCE - Freedom Conversation Engine
FLO, the dynamic conversation engine
GIST
WAnTED - Walk n Talk Engine for Dialogs
ConFlo

## VERSION 1

**step 1:** Build the GUI, just barebones.
gDialog - 1024 x 290
18 buttons on top of it, in 3 columns
column 1: 216 x 32
column 2: 275 x 32
column 3: 512 x 32
button in the bottom-right that opens and closes gDialog

**step 2:** Clicked states.
default button sprites: 5, 6, 7
clicked button sprites: 8, 9, 10
disabled button sprites: 11, 12, 13

*Clicking on an enabled button calls **this.Activate(column)**. so* Button3.Activate(1); *means you clicked on button 3 which is in column 1 and it activates that one, and keeps track of it with global bool **column1clicked**, which also prevents you from activating another button in the same column. The Activate/Deactivate function also enables each subsequent column when a button is clicked. So you click a button in the first column, it highlights that and enables the 2nd column.*

> **step 2.1:** Solve the issue where buttons graphics overlap.
> Since GUI buttons have no Z-order property, only bringtofront and sendtoback, once you bring a selected button forward it will overlap the button from the previous column.
> No (easy) fix.

**step 3:** Resetting the system once a TCU has been completed (aka you click an option in the 3rd column).

*Added the Reset() function which deactivates all columns and returns you to the beginning of formulating a sentence once you complete and speak the previous one. (Also used in opening/closing the GUI to disable all columns.)*

> **step 3.1:** Optimize the reset code by turning the code for every button into a while loop that bulk processes them.
>
> ```
> int i = Button1.ID;
> while (i < (Button1.ID + 18)) {
>  gDialog.Controls[i].AsButton.TextColor = 15;
>  i++;
> }
> ```

**step 4:** Track the buttons you click on.

*Clicking a button stores the button.text in the variable **column#word** (with the magic of the extender function I can use* this.Text*!).*

**step 5:** At the end of the sentence construction, reset the GUI and speak the appropriate line.

*Partly succeeded. I added a **rep_ex check** to the new script **Conversations.asc** which looks at the columnword variables to see which option you selected and then takes the appropriate action. Or at least it should once I ever figure out how in the fuck I'm going to do that...*

> **step 5.1:** AGREE and DISAGREE end in the 2nd column. Implement that exception.
> *Once that rep_ex check was in this was easy. Just check column2clicked instead of column3clicked and then see if the button you clicked was either AGREE or DISAGREE.*

**step 6:** Load a different submenu depending on which option you clicked in the first column.

*Added the **SetTxt(int column)** function which fills a newly activated column's buttons with the appropriate keywords based on which button you clicked in the previous column. (called*

*from ActivateColumn, so it's completely automated.)*
*Also added a check to the ActivateColumn function that only enables buttons that have text. So if you leave button 7's text empty (aka ""), it will stay disabled. ACEWIN!*

**step 7:** Check which characters are in the room and add them to the dialogue GUI.

*Pieced it together using the same while-loop method I used for the inventory items. We check the total number of characters with **Game.CharacterCount**, then loop through all of them using an int and a dummy character, checking to see if they're in the same room as the player, and if they are, place their name on the GUI button.*

> **step 7.1:** store the character you intend to speak to (by clicking on their name in the GUI)
>
> *Done by using again a similar while loop as the one above to loop through all the characters and see if any of their names match column2word (character names can only be found in column 2, so if you clicked on one it shows up in the column2word variable). If so, stores the matching character's SCRIPT NAME in **charID**.*
> *So now you always know which character you last talked to. (but be careful because it could also be null, could cause problems)*
>
> **step 7.2:** actually the biggest problem is that it does not clear when you select YOURSELF or EVERYONE, resulting in the same responses for all those options.
>
> *Fixed (temporarily) by not checking against if charID == cGirl, but if column2word == cGirl.Name. Different syntax, same meaning, better results.*

**step 8:** Add a deselect clause to the Activate function so you can switch between submenus instantly without having to first manually deselect the currently selected button (only applies to column 1 and 2).

*Moved the Deactivate function above the Activate function and added a check to the latter one using the **Button* prevbtn*** variable *that stores the currently selected button and Deactivates it if you click another button in the same column.*

> **step 8.1:** Also close the 3rd column when you switch in the 1st column otherwise it will mess up because it doesn't change along.
>
> *Fixed by adding a DeactivateColumn(3) to the Deactivate function for column 1.*
>
> **step 8.2:** The font color of any selected buttons in column 2 and 3 doesn't change back.
>
> *Forgot to add a **TextColor** reset to the DeactivateColumn function. Fixed.*
>
> **step 8.3:** After you select something in the 2nd column and go back to the 1st column it is possible to select TWO options. Woops.
>
> *Caused by not differentiating which column the prevbtn belongs two, seeing as it overwrites the button from the 1st column if you click anything in the 2nd column. Fixed by splitting them up into prevbtn1 and prevbtn2.*

**step 9:** Take another jab at trying to figure out how to parse the appropriate response to whichever option you clicked.

**step 9.1:** Divide dialogue responses per character?
Sectioned the responses per charID in the rep_ex of Conversations.asc, creating [cases] for each character instead of putting it all in the script of the matching button.

**step 9.2:** Find a way to disable/mark options you already clicked.
Construct some sort of default master list of all the options, because ActivateColumn resets the buttons every time, regardless of whether you disabled them or cleared the txt.

*(Presumably) fixed by storing the options in a variable instead of applying them directly to the button texts. These variables (**btn[10]**.etc) are declared by a **struct** at the top of Dialog.asc. This way we have a **catalog** of all the options, which you can edit at any time. The ActivateColumn function reads from this struct to fill the options.*

*Now that I think about it, we could create a separate catalog for each character too!*

**step 9.3:** Take option definitions out of setTxt function so you can modify them ingame without setTxt constantly overwriting them with its default list.

*Created a **struct** that stores the dialog options. But not sequentially. Rather, it uses the struct index as a matrix! so dlg[2102] would be NPC 2, ASK, option 2. It leaves a lot of indices unused but whatever. It has a .Text variable for the button text, .Enabled, and .Response. Moved the struct definitions into **Catalog.asc**.*

**step 9.4:** Turn the .Response attribute of the struct into a function.

*Apparently you can also define functions as part of a struct, so I turned .Response into **import function Response()**, which THANK GOD I can define in the Responses.asc so I can keep it separate from Dialog.asc and Catalog.asc. The syntax is really weird with stuff like dialogue::Response, but it works.*

**step 9.4.5:** Find a way to get the current index of the array.

*Added a very vague function to the struct courtesy of Monkey_05_06 that apparently somehow stores the ID of the array index you currently selected. I'm not sure how it works but apparently it does, because from inside the Response function I can now call **this.ID** and it will give me the index of the option I just clicked!*

**step 9.5:** Reorganised the script to move the responses out of the rep_ex and into the Response function. Response is now triggered from the rep_ex by comparing the button text to the struct Texts in an overly complicated while loop (but it works). Right now it's triggered by column3clicked, but that excludes AGREE and DISAGREE. I'd really rather not put them in the rep_ex also, so find a way to check for those two in that complicated while loop.

Added an extra clause to the rep_ex check for AGREE and DISAGREE (as well as a few dozen other places that were apparently now broken because of the struct), and

the actual response thread is in the Response function.

**step 9.6:** Switching between categories is broken, null pointer error!

*This was an accident waiting to happen which apparently lay dormant before the introduction of the struct. The problem was that DeactivateColumn function kills column 3 as well if you switch categories in column 1, but if you never clicked anything in column 2 (which would populate column 3), it just runs into a lot of nulls for the button texts. Fixed by adding an if (column2clicked == true) check to the column 1 deactivate section.*

**step 9.7:** Replace the "" buttondisablecheck with a check for .Enabled on the struct entries themselves.

*Can't because it would have to happen in the Activate function and I can't query which dlg options are used there (not unless I rewrite the WHOLE thing).*
*On hold for now.*

**step 9.8:** Add another digit to the array to keep track of the sub-submenu in RESPOND?

*There will never be response options in the double digits, so use those to map the third column options on, like 0210 is AGREE and 0224 is JOKE.*

**step 9.9:** Somehow the parser stops working after the second option you click in the same column as the previous option. Where does this stem from??

*resetting the charID to null at the end of the Response function broke the system, because it relies on charID to formulate the index, and since you clear it every time it will return the wrong value for all subsequent options until you re-open that submenu. But if I don't clear the charID it breaks all non-character related interactions.*

*To solve this I remapped YOURSELF to the player character slot (0), and moved the **charID = null** definition into the Respond section in the SetText function.*

**step 10:** Trigger a timer in the Reset function to measure idle time between responses.

*Timer **20** was used. Clicking on a button in column 3 cancels this timer, because by then a sentence is constructed and ready to be spoken. Referenced in the rep_ex_always of Conversations.asc to trigger idle responses.*

**step 11:** Add a label under the dialog GUI that notifies you when options are changed.

***INotif** was added, with the function **Notify(String text)** in Dialog.asc, which fills this label with text and initializes a timer (19) to clear it again after 4 seconds. Used when for instance you pick up an item and it is added to your 'vocabulary'. Also used to keep track of NPC disposition now.*

**step 12:** Reinstate the idle timer and have it affect NPC responses

*Had to do a little magic to get the idle timer to work correctly again, but it works now - once*

***timer 20*** *is expired, the rep_ex in Responses.asc handles idle responses, which should play every time the charID is not null (and also not yourself). Also added the* ***int ignores*** *to Dialog.asc, which keeps track of the number of consecutive times the idle response is triggered. In this case the NPC leaves after 3 'ignores' (which is the 4th time it is triggered, mind you).*
*ignores is reset in the Response function (so every time you click on an option in the GUI).*

## VERSION 2

**step 1:** Tightened up the vertical spacing of the options to create more room.

**step 2:** Investigate the possibility of having scrolling columns to accommodate more options. Use the same amount of buttons but scroll through the array with mouseup-down???

*Too hard.*

**step 3:** Multiple characters breaks the GUI, null errors whaaaat!

*Turns out I didn't specify in the SetTxt what should happen with a charID above 0 or 1. I didn't want to have to fill it out for every character, so I made it modular by using GetIndex with the charID to find the appropriate index range matching the character you'd wanna talk to, and then loading that up on the GUI. Sounds complicated, works great.*

**step 4:** items added to the dialog don't show until you reload the GUI - an Update function?

*For the time being I stuck a Reset(); into Notify to make sure the options show up. Not the best solution but it works.*

**step 5:** Removed the GIVE > PAT ON BACK business and just added the inventory items everywhere. Also renamed GIVE to SHOW, making a separate inventory all-but redundant. The only thing that's missing is item combining. Examining items can now be done by SHOW > YOURSELF.
Also removed the POLITE and BLUNT submenus and replaced them with POSTIVE and NEGATIVE, context-sensitive actions. These also replace THUMBS UP & DOWN. And in between them I added PLAYFUL to make up for cutting JOKE.

**step 6:** Added an arrow to select button graphics to indicate that option has a submenu.