# CS 369 Computer Game Design, January 2011

Dr. Bruce A. Maxwell
Department of Computer Science
Colby College

**Course Description**

This course covers the design and implementation of 2-D computer games using a commercial game engine. Topics include game design, artistic concepts, image manipulation, game scripting, and basic artificial intelligence algorithms. Students will work in groups that mix computer science majors and non-majors to design a 2-D game to be distributed at the end of the term

**Prerequisites:** CS 231 or permission of instructor.

# 1 Games

What is a game? Why do we play games? What makes them a desirable (fun) activity?

Games are a form of entertainment, but they are more than sitting reading a book or watching a movie. Games involve interaction and activity. They are really a form of creative expression, not just entertainment. Games are not random play, however. Games have a point, or a challenge to them.

A game lives in its own world, often called the magic circle, and the purpose of the game is to overcome the challenge within the world of the game. The world of the game has its own, self-consistent and complete rule set. All possible actions and states encountered within a game must be defined by the rule set.

Sometimes the challenge in a game is relative to another player. But if the players do not interact during the activity, is it still a game? A game must allow interaction between players, or the activity becomes a competition, not a game. In fact, a game must have some type of conflict where one agent is attempting to block another.

Crawford has a useful diagram in his *On Game Design* book that uses a tree of concepts to separate games from other forms of activity. To differentiate various forms of creative expression, he asks five questions about characteristics of the activity.

| For Money? | Interactive? | Goals? | Competitor? | Attacks? | Category |
|---|---|---|---|---|---|
| No | No | No | No | No | Art |
| Yes | No | No | No | No | Entertainment |
| Yes | Yes | No | No | No | Toys |
| Yes | Yes | Yes | No | No | Puzzles |
| Yes | Yes | Yes | Yes | No | Competitions |
| Yes | Yes | Yes | Yes | Yes | Games |

Many programs we call computer games are not actually games, according to Crawford. SimCity, for example, is a Toy in some play modes, but would probably fall into the category of a Puzzle. Many computer games are race games where competitors cannot impede one another. Small modifications to competitions, however, can turn them into games, such as allowing race cars to bump into one another.

Crawford: "a game is a closed formal system that subjectively represents a subset of reality."

- The 'closed formal system' means the game has explicit rules that define the set of actions and states within the game.
- A game has both an objective reality–chits, cards, figures, dice, a board, icons–and a subjective reality–aliens, marines, cities, carriers, people, houses, monsters. The objective reality provides concrete indicators of the game state, while the subjective reality is a fantasy world built on top of the objective game state. The degree of difference between the objective and subjective realities does not actually limit the degree of fun in a game (consider chess, or the original D&D play modes). Modern computer games, however, have significantly reduced the visual, aural, tactical, and even proprioceptive differences between reality and fantasy. A good game will make the subjective reality easy to access and enjoyable or challenging.
- A game is not reality. A game does not need to follow the rules of reality and, in fact, may be more fun if the rules of the game flaunt or otherwise ignore social customs or even physics. A game must balance the complexity required for gameplay with the need to keep the game focused and limited in scope. A game with too little focus becomes open-ended and loses the challenge that makes it fun.

## 1.1 Game Types: Genres

Creating a taxonomy of game types is not a trivial task. The various categories of games are generally called *genres*. We can define genres objectively, but many genres reflect a lead successful game that offered something new that was then copied and modified by successive games within the new genre. Games can be divided by many different variables, and a single game may contain multiple values of a single variable–putting it in several different genres–especially if the game involves a series of different kinds of challenges.

- Input device: keyboard, mouse, joystick, gamepad, specialized input device, wireless device, wii
- Number of players: single player, multi-player, massively multi-player
- Iconography: science-fiction, fantasy, card games, realism, abstract, ...
- Speed of play: turn-based, continuous action, simulation with time control
- Number of actively controlled units: one, small group, large units and formations
- Source of challenge: physical speed, reflexes, problem solving, multi-tasking, strategy
- Level of Control: player controls all action (e.g. card games), player only indirectly controls action (e.g. The Sims).
- Point of View: first-person, third-person, overhead, 2D, 2.5D, 3D
- Purpose: education, mental stimulation, enjoyment, training, therapy
- Richness of environment and environmental interactions: simple, linear, free-form
- Goal: achieve a state of harmony, kill the bad guys, self-improvement, beat the competition, education

There are a number of different taxonomies of video games, and new games have blurred the line between different types. For example, Wolf (Chapter 6, "Genre and the Video Game" in *The Medium of the Video Game*, Wolf ed., 2002), proposes a more specific, horizontal taxonomy, where many (if not all) games belong in more than one class.

Wolf Proposes the following categories:

| Category | Description | Example |
|---|---|---|
| Abstract | Non-representational graphics, not much narrative | *Tetris* |
| Adaptation | Adaptation of a game from another medium or activity | *Price is Right* |
| Adventure | Complex interactions with a rich environment | *Raiders of the Lost Ark* |
| Artificial Life | Simulation of creatures requiring input from the user to achieve some goal state | *The Sims* |
| Board Games | Video game adaptations of board games | *Chess* |
| Capturing | Catch stuff that runs away | *Gopher* |
| Card Games | Video game adaptations of card games | *Bridge* |
| Catching | Catching stuff that may move, but does not actively run away | *Big Bird's Egg Catch* |
| Chase | Pairs with Catching, Capturing, Driving, Escape, Flying, Racing | |
| Collecting | Collecting things that don't move | *Pac-man* |
| Combat | Equally matched projectile-shooting entities | *Battlezone* |
| Demo | Game demonstration, usually not fully-featured | |
| Diagnostic | Diagnostic programs (not really games) | |
| Dodging | Primary objective is avoiding projectiles or moving objects | *Frogger* |
| Driving | Games based primarily on driving skills | *Pole Position* |
| Educational | Games designed to teach | *Mario Teaches Typing* |
| Escape | Objective is escape from pursuers or a maze | *Ms. Pac-Man* |
| Fighting | Hand-to-hand combat | *Mortal Kombat* |
| Flying | Games that involve flying skills | *Microsoft Flight Simulator* |
| Gambling | Adaptations of gambling games | *Blackjack* |
| Interactive Movie | Branching video clips | *Star Trek Borg* |
| Management Simulation | Resource management games | *SimCity* |
| Maze | Objective is successful navigation of a maze | *Lode Runner* |
| Obstacle Course | Traversing a set of obstacles | *Boot Camp* |
| Pencil-and-Paper Games | Games usually played on paper | *Hangman* |
| Pinball | Self-explanatory | *Pinball Wizard* |
| Platform | Moving through a series of levels and avoiding obstacles | *Donkey Kong* |
| Programming Games | Player writes short programs to control agents | *CoreWar* |
| Puzzle | Primary objective is to solve a problem or a riddle | *Myst* |
| Quiz | Tests player's ability to answer questions | *Trivial Pursuit* |
| Racing | Primary objective is to win a race | *Pole Position* |
| Rhythm and Dance | Objective is to keep time with and respond to music | *Guitar Hero* |
| Role-Playing | Player takes on the persona of the character | *Diablo* |
| Shoot 'Em Up | Shooting lots of opponents or objects (one-sided battle) | *Space Invaders* |
| Simulation | Meta-category for Management and Training Simulation | |
| Sports | Adaptations of sports | *Madden NFL 2007* |
| Strategy | Primary skill required for success is strategy; often turn-based games | *Spaceward Ho!* |
| Table-Top Games | Adaptations of table-top games | *Pong* |
| Target | Primary objective is hitting a target | *Shooting Gallery* |
| Text Adventure | Adventure/role-playing game with a text-based interface | *Zork* |
| Training Simulation | Simulation of realistic situations | *Flight simulators* |
| Utility | Programs with functionality beyond games | |

Obviously, one could develop alternative taxonomies, and perhaps a hierarchical taxonomy that divides along a sequence of variables might offer more interesting insight. After all, the purpose of a taxonomy is to create an abstract representation of a complex set of data in order to permit meta-reasoning and understanding of complexity.

## 1.2 What makes a good game?

We can divide games into genres along various different axes, but just because a game is in a particular genre does not mean it is fun to play (even if you generally like the genre). You might really enjoy SimCity, play SimTower every once in a while, but SimFarm is pretty boring (IMO).

What are the factors that make gameplay good, or fun?

Three basic concepts to consider are identified by by Salen and Zimmerman:

1. Meaningful play: actions of the player have meaning by how they affect the outcome of the game.
2. Discernable effects: the effects of actions (or inaction) must be discernable to the user.
3. Integrated effects: player actions should have not only an immediate effect, but a long-term effect on the game outcome.

Here's my list of generic things that make for good gameplay:

- It has to be challenging (there has to be a reasonable chance of failure).
- You should be able to improve with play.
- You should always feel like you can win if you try harder, practice, and/or make the right decisions.
- There should be intermediate rewards.
- The game should be consistent within itself.
- It should feel like your opponents are playing fair.
- The game should offer complete functionality within the format of the game world.
- The interface should be usable with a small amount of practice.
- The interface should make all information required for successful gameplay easily visible to the user.
- All actions by the user should impact the outcome and larger context of the game.

There are also genre-specific issues, which is where things like ambience, visual effects, expected capabilities, and player interaction probably fit.

One goal of video games, which is enabled by the visual aspect of the medium, is putting the player in a zone where the game is their perceptual world. What are the characteristics of a game that enable this effect?

- Quality/smoothness of the interface: does it fit naturally with the game actions?
- Time: a feeling of time pressure by the player contributes to the zone (whether or not it exists).
- Engagement: the game process has to be interesting and achievement desirable.

# 2 Game Design

The fundamental elements of a game often have little to do with the physical aspects of the game. The required aspects of a game are the following.

- A starting condition: what is the initial state of the game world?
- An end condition: what are the end states of the game world?
- A set of rules: what actions are available to the player at each state in the game world?
- Any items or capabilities required to follow the rules.

Challenge arises from the difficulty of achieving a good end condition given the starting condition.

For some games, physical objects are a necessary component of the game (e.g. ultimate frisbee, paintball, or laser tag). For many games, however, the physical objects are simply indicators of the current game state. Monopoly, for example, does not require a board, figures, or money; these physical items are only used as a convenient representation of the current game state. The only item required by Monopoly is a mechanism for generating random numbers according to the sum of two uniform distributions over integers in the range [1, 6]. Two six-sided dice are convenient, but not required. Six slips of paper with numbers written on them would accomplish the same goal.

- The board is a connected graph, with links from each location to the jail square.
- Each player is representable by a link to a location on the graph.
- Each player maintains an integer representing current cash on hand.
- Each player maintains an array of properties, which are data records.
- The chance and community chest cards are each an array of records that can be accessed in random order.
- Each board location is a record indicating its status, or value (free parking).

The mechanics of a game are the end result of a game design process.

Crawford suggests picking a goal first, where the goal is not the goal within the game world. Crawford's meaning of goal is what the game designer wants the player to experience or learn through the act of playing the game. The meta-goal of the game does not necessarily have anything to do with the gameplay.

The second step in game design is to pick a topic, which is the overall context in which the game world resides. Will your game context involve aliens on another planet, or will it take place on a table-top with spoons and forks as the actors? The topic should support the overall goal of the game, but there is wide latitude in picking the topic.

The concrete design phase begins with decisions about the mechanics. Here, the genre of the game begins to suggest the form of the interface, number of players, and the overall structure of the game.

- Is the game multi-player?
- Is the game action, or strategy?
- Is the game real-time or turn-based?
- What are the set of actions the user can make?

These decisions largely determine all of the necessary interface elements. If the game is multi-player, then the user will need screens enabling them to connect to a server. If the game is an action game, the player will need to be able to move the character and engage in attacks (generically defined). If the game is strategy, the player will need to be able to move pieces or agents.

The same variables also determine what information the computer must display to the user. Revealing only part of the actual game state is one way to make a game more challenging. But there are certain variables a user must be able to access–amount of resources in a strategy game, or health in an FPS–in order to make proper decisions and achieve the goals of the game.

Given an interface and general outline of the game play, the next step is to make concrete decisions about game play. What are the actual set of actions the user can take? What actions can the opponents take? How fast does the game move?

Then next step, for computer games, is to begin to plan the actual program design.

- What classes or objects do you need?
- What information is required for each class, object, or agent?
- How will you make the opponent (assuming it is a computer opponent) select actions?
- What events in the game play need to trigger actions?

The final steps are to develop the art and interface and then to fine tune the game play. How much should a sword cost? How much should armor cost? How many health points should each player have? How fast can the bad guys move? These are all important questions, but should be addressed once the basic mechanics of the game are complete. Only then can you actually make decisions about how they impact gameplay.

# 3 Intellectual Property and Copyright

The importance of intellectual property in video game design cannot be understated. The video game industry is a mutli-billion dollar industry and relies almost completely on intellectual property rules as the basis for earning money. The flow of information is incredibly fast, and there are so many people looking at games that you can be sure any violation of copyright will be quickly identified. You want to make sure that anything you put in your game is original work.

**Copyright** is the most important aspect of intellectual property with regard to video games, although trademarks are also relevant for names and phrases.

Copyright applies automatically to original works of art, writing, and code. Copyright also covers the right to create derivative works, so tread carefully when using ideas from novels, books, or movies. You do not need to do anything formal for your own work to be copyrighted; it is copyrighted automatically. However, you probably do want to note dates of creation and authorship of the elements of your game in case there is ever a question. Start every code file with 'copyright <year> by <author>'.

All material from all video games ever created is copyrighted. Some material may have been explicitly released into the public domain, but none of the copyrights have expired. Copyrights last for the life of the author plus 70 years, or for 95 years from publication/120 years from creation in the case of works for hire (whichever is shorter). Because of an act of Congress in 1998 that extended copyright by 20 years, no materials will enter the public domain until 2018.

## 3.1 Copyright: Fair Use

In video games there is no fair use unless you are creating your video game for personal reasons with no intention of sale or profit. Fair use depends upon four factors, most of which work against using copyrighted material in video games:

1. The purpose and character of the use, including whether such use is of a commercial nature or is for nonprofit educational purposes
2. The nature of the copyrighted work - highly personal works get more protection
3. The amount and substantiality of the portion used in relation to the copyrighted work as a whole
4. The effect of the use upon the potential market for or value of the copyrighted work

The bottom line is that you want to take your own pictures, avoid using trademarked items without permission, and don't just grab stuff off the internet without thinking about what you're doing.

I'm working under the assumption that Colby makes no claim to student intellectual property generated as part of a course. Therefore, your group needs to make a decision about how IP will be handled.

As the development of a video game is both a group activity and an activity with the potential for profit, it is important for each group to develop an intellectual property agreement that treats fairly all members of your group as well as any other person who provides IP used in the game. Your agreement may be as simple as the following one line.

*The ownership of all intellectual property developed for the CS 369 Computer Game Design course taught in January 2011 by the group consisting of [your names here] will be shared equally by all parties.*

# 4 Game Planning and Design

**Realistic goals (Kanade rule)**: If you think you can finish a project in X units of time, double X and increase the units of time by an order of magnitude. *For a 1-month project, pick something you think you could finish in 1/2 of a week if you had nothing else to do.*

Part of setting realistic goals is limiting the space of possibilities. In any design project, it is always possible to make the design better or add more features if you have more time. But time is not an infinite quantity. Limiting your game design to fit within certain constraints can also encourage more creativity. It can be difficult to be creative in the absence of constraints. Reasonable constraints force us to be creative.

The following are some game parameters you can constrain to limit the complexity of the design.

- Game purpose: pick something that interests you.
- Game skill: what does the player have to learn to be good at?
- Genre: will your game conform to a specific genre type?
- Expected time to complete a level of the game: quick game or long?
- Number of players: one, two, or many.
- Narrative quality: is the game a linear story, branching, or free-form?
- Number of levels: one, N, or infinite?
- Number of active elements in the game: two, five, ten?
- Restrictions on player actions: can only move in one direction, can only fire in one direction.
- Restrictions on the user interface: can only see within a limited window.

**Iterative design concept**: Create (physically) a rough interactive prototype as quickly as possible with people playing people. Make the game out of paper and soda straws if you have to, but start with an initial set of rules and players and see if the ideas work.

**Salen and Zimmerman Rule**: 20% of the way into the game design, have a prototype.

## 4.1 Game Design Process Summary (Crawford)

1. Goal and topic development: figure out what you want your game to do. Think about the purpose of your game, and make it something you care about. Then select a topic that is appropriate.
2. Research and preparation: do your homework on your topic to get big picture and the details right.
3. Design phase: develop the overall game concept.
   - User interface design: how does information flow to and from the user?
   - Game structure: what are the key elements of the game? Where does strategy or decision-making enter the game? What are the options of the player with respect to the key element? The game must balance expressiveness and limiting choices to permit smooth continuous action.
   - Program structure: what are the key structures required by the interface and the game structure? How do they communicate? How is the data organized?
4. Evaluation of the Design (go-ahead decision is here): play the game as a prototype version on pencil and paper, or with little figures. Consider stability and balance. Does the game achieve your goals?
5. Pre-programming phase: documentation of the game design.
6. Programming phase: implementation of the game design, with lots of iterations.
7. Playtesting phase: testing of the game implementation, with lots of iterations.
8. Post-mortem: how is your game?

## 4.2 Developer Roles

**Producer:** The producer is the overall organizer and manager of the project. This person does not need any special expertise, but some knowledge of all the areas of game design is important. The producer is the one who watches the calendar and provides motivation where needed, and assistance when required, perhaps by diverting resources from one part of the project to another. In a real game development situation, the producer also handles most of the outside contact and money issues.

**Designer:** The game designer develops the rules, concept, goals, interface, and the various and sundry other pieces of a game. The designer is good at generating, testing, and weeding out ideas. Most members of the game development team play some role as a game designer. The lead designer is responsible for integration of the ideas into a playable game.

**Programmer:** The game programmer puts the design concepts into practice. The challenge is figuring out how to implement the ideas put forward by the design team.

**Visual Artist:** The visual artist develops the visual content for the game. Most game engines use abstraction to separate the specifics of the visual art from the game programming so updating visual art is easy.

**Audio Artist:** The audio artist develops the audio content for the game.

**Quality Assurance Specialist:** The QA person is in charge of testing and evaluating the game as it develops. Most members of the team will take part in QA, but the lead QA person is responsible for ensuring that all aspects of the game have been tested.

## 4.3 Planning Tools

**Critical Path Management**: Critical path management [CPM] is a technique for identifying bottlenecks in the development process.

- List each task in the design process, dividing the process into small, but natural chunks of work. Each task should have a clear outcome or product.
- For each task, estimate how long the task will take, list the inputs to the task and list the output of the task.
- Arrange the tasks along a timeline, showing the relationships between inputs and outputs. Start at the end and work backwards. Place tasks in parallel wherever possible.

CPM can help you identify critical tasks along the development path. Often, several parallel streams will converge, and work cannot continue until a single task is complete. Dividing tasks into pieces also helps you estimate the amount of time a task or project will take.

**Versioning Control System**: Set up something (e.g. cvs) so that you have a versioning system working that manages the issues of multiple people working on the same project. It is quite reasonable to make a CVS repository for all of your Torque scripts.

## 4.4 Integration

Integration always takes longer than you think. If you have people working on parallel tracks, plan for at least 20-40% of your overall time to be integration of the pieces. Testing is a necessary part of integration because there will be problems in code, etc., that don‘t show up until things are integrated. One of the biggest challenges with Torque is maintaining name spaces properly. For example, if two design teams use the same name for different player, button or GUI elements, that will cause problems when the system is integrated.

# 5 Designing for an Audience

*Author's note: need some more general guidelines about designing to an audience here. Basically, know your audience and do some research to figure out what they like.*

Question: who is your audience?

> *If we do not regularly state that a percentage of our audience is expected to be female, we assume we are designing for males.*
> – Sheri Graner Ray, Gender Inclusive Game Design: Expanding the Market, 2004.

Most of the myths of designing neutral or equitable gender games have been shown to be false.

**Myth 1** Males won't purchase games with female lead protagonists.

In 1989, one of the first game companies founded by a woman was hesitant to have a lead female character, because they didn't want to lose their male audience. However, the first game they sold with female protagonists–*King's Quest IV*–had even better sales.

*Tomb Raider* likewise shattered the myth. Sony enjoyed huge sales and made the game part of the playstation rollout. Sega and Nintendo had refused to build similar female lead characters, believing it would threaten their main market: young men.

**Myth 2** Females don't enjoy video games as much as males.

In 1999, *The Sims* completely dismissed the myth that women don't play video games

- *The Sims* became the biggest selling title of all time soon after its release
- It is estimated that over 60% of Sims players are women
- *The Sims* was subsequently beaten by *The Sims II* and *The Urbz: Sims in the City*

**Myth 3** Females are less skilled at computer usages than males

Research has shown this to be false.

- Equal exposure to computer games decreases pre-existing gender differences (Greenfield, 1996)
- Given equal access to computers, there are no gender differences in programming ability (Linn, 1985).
- A generally more positive attitude and more experience with computers among males does not translate into higher performance in computer courses (Woodrow, 1993).

Conclusion: There is a big market out there that includes men and women in equal measure.

## 5.1 Gender considerations in design

Research shows there are gender differences in the way games are played and designed.

In one study (Van Eck, 2006), when teams of all boys, all girls, or mixed genders designed games, both tended to make adventure games centered around exploration.

- Boys tended to make conflict the central element of the game
- Girls tended to avoid direct conflict as a solution and required players to use other options

In the same study, when boys and girls played the same game, *Sim Safari*, they enjoyed it equally well, but they used different parts of the game.

- Girls designed high-end, detailed dwellings
- Boys designed swamps, crocodiles, and jaguars.

A second study (Hartmann, 2006) surveyed German women to determine their impression of a set of fake video games designed to evaluate three factors: social interaction, violence, and physical stereotypes. The average responses rated opportunities for social interaction the most important factor, followed by a non-sexualized role for the female protagonist, and then non-aggressive content. Almost half (44% of respondents) reversed at least one of the factors with respect to the average order, although the social interaction aspect was generally not reversed with respect to the other two.

In the same study, male and female respondents differed in their intensity of gameplay based on competitiveness. In non-competitive genres, no difference was observed.

Why do we care?

Historical facts:

- Up through 1991 92% of arcade games contained no female roles; 2% had active female roles.
- Less than 10% of the audience for traditional PC games is female
- Less than 15% of Nintendo's user base was female before the wii
- Less than 20% of the audience for traditional online titles are female
- 70% of casual, online gamers are female (44% overall online gamers, (ESA, 2005))
- 52% of internet users are female
- The video game industry had over $10.5 billion in U.S. sales in 2005 and reached $20 billion in sales in 2010 (excludes hardware sales)
- As of 2005, in the US, 43% of all video game players were female (ESA, 2005))

There is clearly a place, and money to be made, for games that are gender neutral.

But building games with gender in mind does not mean building around stereotypes. Initial attempts at building games for women failed miserably.

> *The industry took an entire market of women and defined it as a genre of "fashion, shopping, and makeup games for girls ages 6-10."*
> – Sheri Graner Ray, Gender Inclusive Game Design: Expanding the Market, 2004.

One of the exceptions was Mattel's *Interactive Barbie*, which was more of an interactive design space and in many ways an accessory for physical Barbie dolls.

Gender considerations in design do not mean stereotyping the audience. Instead, it has more to do with balanced design and subtle genre considerations. One interesting observation is that gender stereotypes, in some ways, more strongly affect men than women. The relevance to game design is that to design games women will find enjoyable, we really don't have to go out of our way to actively design elements of the game to appeal to women; **we just have to avoid designing our game solely to fit male stereotypes**.

Some of the guidelines proposed by Shei Graner Ray, (an academic, and founder of a game company focused on games for women), include the following (which are clearly some broad generalizations):

- Learning styles: Males tend to more explorative and risk-taking. Females want to know how something works first and tend to model or imitate as part of learning.
  - Most tutorials in current games are explorative in nature.
  - Design: generate tutorials with imitative models as well as explorative models
  - The learning style difference is not necessarily limited to gender, or explorative v. imitative. As a designer, consider visual, audio, and haptic learning differences when building tutorials.
  - **However: the most successful cross-gender games are exploratory in nature, so there is a separation between learning a game and playing a game.** (Van Eck, 2006)
- Price of failure: Males expect punishment for error, females forgiveness (very broad generalization).
  - Design: design victory conditions such that failure does not result in irretrievable loss.
- Avatars: since they often represent heroic characters, avatars normally exhibit exaggerated physical characteristics. Female avatars are often given exaggerated sexual characteristics, while male avatars are given exaggerated size and muscles. Whether these constitute sexual characteristics is arguable, but certain male sexual characteristics are generally not exaggerated.
  - Consider that culturally, it is more difficult for males to be "sissies" than females to be tomboys.
  - Design female avatars based on female athletes (avoid hyper-sexualization)
  - Focus group test all avatars with female characters (don't assume anything)
- Communication: Males and females tend to communicate differently, even in electronic media. In particular, there is gender preference for formality and rapport-building language.
  - Avoid industry or genre specific jargon in documentation, tutorials, and game scripts (e.g. WASD).
  - Avoid using content containing sexual humor and negative comments.
  - Be inclusive about including formality and rapport building language in commands
  - Real social interaction is interesting.
- Production environment: who you are defines, in part, what you design.
  - Avoid "he" in all documentation and tutorials. Do not assume your player is a male.
  - Include women in the design and testing process.
  - Clearly state who constitutes your intended audience. If your audience does not include females, you are cutting out half of your potential market share.

# 6 CVS Notes

The Concurrent Versioning System [CVS] is a simple system for sharing files among multiple people. There are a number of versioning systems available, with some commonly used systems being cvs, svn, and git. The latter two are newer and have some more advanced features. However, for small to medium-size projects CVS offers a simple interface and is easy to set up and manage.

The main idea of all versioning systems is that there is a single master repository. Users **check out** their own copy of the repository when they need to work on the project and periodically **commit** their changes. Users regularly **update** their own repositories with changes others have made to the master. When a prior change made to the master conflicts with a local change on an update, it is the user's responsibility to resolve the conflict and commit the resolved file.

CVS uses a single master repository, and all commits affect it. More recent versioning systems, such as git, permit users to create branches–repositories separate from the master–and to commit changes locally so they don't affect the master repository until the users specifically pushes the changes.

**Setting up a repository**

Your group needs to set up a repository only once. Put this in a file space everyone can access (read and write) and that is regularly backed up.

```
cvs -d <path> init
```

You should see a directory called `CVSROOT` appear in your selected repository.

**Importing a project**

Your repository is initially empty. Build your project locally. In the case of Torque, create an empty project and create a level file. Then cd to the top level directory of your project and import the project into the CVS repository.

```
cvs -d <path> init -e nano import <repository name> <project name> initial
```

A new directory with your repository name should now exist in your CVS repository.

**Checking out**

When you want to work on your project for the first time on a particular computer (or file system) you need to check out the repository. This will create a local copy of the repository that you can edit.

```
cvs -d <path> checkout <repository name>
```

You should now see a new directory in your current directory with the repository name. Each directory within it should have a directory titled `CVS`.

**Updating**

If you have already checked out a repository and want to just get the latest changes, you need to update your local copy. You should do this often, at least once per day when you start to work. You should be at the top level of your checked out project in order to get all of the updates.

```
cvs update
```

Note that the standard update does not give you new directories within the project. To get new directories, add the `-d` option at the end `cvs update -d`.

### Adding

When you have created a new directory or a new file you need to add it to the repository. From the directory to which you added the items, use the following command to set up the files or directories for addition. Note that if you add a directory it does not add any of the files inside the directory. You must add those individually.

```
cvs add <list of files or directories to add>
```

### Committing your work

When you have made changes to the project, and those changes are ready to be pushed to the other members of the group (no one likes to get updates that don't work), then you need to commit your work to the master repository. First, execute an update so that you have the latest changes from everyone else. Then double-check everything still works properly. Finally, commit your updates. From your top level directory, use the following command.

```
cvs commit
```

Note that you can commit a single file or subdirectory if you wish. If, for example, you are in a sub-directory of your project, only those files at or below your current subdirectory will be committed.

When you commit, CVS wants you to put a log message into the system indicating what you have done. This is generally a good idea. You can add messages in one of two ways. First, if you just type `cvs commit`, it will put you into an editor (vi by default) where you can type a message. You can modify the editor by using the `-e` flag. For example, the following will put you into nano.

```
cvs -e nano commit
```

You can also put your message on the command line. The following will put the log message in quotes after the `-m` flag, and it will not put you into an editor.

```
cvs commit -m "this is a log message"
```

### Conflicts

Most of the time CVS is able to adequately merge the master repository with your work. However, sometimes when you update your repository CVS is unable to figure out conflicts. Conflicts can arise, for example, when you and someone else have edited the same section of code in the same file. CVS will report conflicts when you update.

Each file that CVS is updating from the master repository will get a letter designation during update. A `P` means that your version was old and CVS is completely replacing it from the master. A `M` means CVS is merging the file from the master with your file, and a `C` means there is a conflict.

CVS marks conflicts in a file with a series of less-than and greater-than symbols. Files with conflicts contain both the code from your version and the code from the master version. It is your responsibility to go through and fix the conflict, usually by deleting one or the other of the sections. After repairing a conflict, double-check that all of your code runs properly and then commit.

Note that CVS will not let you commit if you do not have the latest updates.