Unit-1 Game Core Design & Initial Design

Deptt. of Info. Tech.
KDK College of Engineering Nagpur

Objectives

- Know the essential elements of a game—rules, goals, play, and pretending—and what they do in the context of playing games
- Know the formal definition of a game
- Understand the nature of challenges and actions, as well as the formal definition of gameplay

What Is a Game?

- Toys do not have rules or goals
- Puzzles have goals
- Games have rules and goals

What Is a Game? (Cont.)

- A game must include
  - Play activity
  - Pretended reality
  - A nontrivial goal
  - Rules

Differences between movies and a game

Sr. No.	game (programming)	movies (animation)
1.	requires any input from a user	a movie (animation) does not
2.	Players have to make decisions	do not need to make any decision
3.	have no ending, sometimes have one ending, sometimes have multiple ways to end	a movies has a predetermined ending
4.	a game has rules and goals, and may have constrains, players, objects, terrains and behaviors, for example, physics, sound, speech, and emotions	a movies has characters ,role & audio, video etc.

Essential Elements of a Game

- Play
  - Play requires participation
  - Making different choices while playing the game a second time affects the results
- Pretending
  - Creates an artificial reality known as the *magic circle*
  - Artificial importance is assigned to events within the magic circle
  - To leave the magic circle, stop playing the game

Essential Elements of a Game (Cont.)

- Goal
  - Every game must have a nontrivial goal or object
  - The rules define the goal
  - The game designer sets the rules, thus defining the object of the game
  - The player must overcome one or more challenges to achieve the goal
    - The goal is often a *victory condition*, but victory or defeat is not required in all games

Essential Elements of a Game (Cont.)

- Rules
  - Rules are definitions and instructions that players accept for the game
  - Rules define the actions the players may select that will help them achieve the object of the game
  - Game designers must make the rules understandable to the player

Things That a Game Is Not

- A game does not have to include
  - Competition
  - Conflict
  - Entertainment
  - Fun
- These are qualities of some games, but not essential to the definition
- Serious games are not necessarily made for entertainment or fun

Gameplay

- The player must overcome a nontrivial challenge
  - Challenges require mental or physical effort
  - A challenge can be composed of several smaller challenges
- Challenges can be required to reach the goal or optional to add game content

Gameplay (Cont.)

- The rules determine what actions are available to the player(s)
- Different actions may be
  - Permitted by the rules, or
  - Required by the rules, or
  - Prohibited by the rules
- Video games permit only actions that are programmed into the game

Gameplay (Cont.)

- Gameplay therefore consists of:
  - The challenges that a player must face to arrive at the object of the game, and
  - The actions that the player is permitted to take to address those challenges *plus* other possible actions that are enjoyable

Fairness

- Players expect that the rules will guarantee that the game is fair
- Fairness is not an essential element of a game, but a quality of good games
  - Players often change rules of a game if they perceive it as unfair
  - Fairness is particularly important in video games because video games seldom allow rule changes

Symmetry and Asymmetry

- In a symmetric game, all players use the same rules to accomplish the same goal
- In an asymmetric game, different players follow different rules to accomplish different goals

Symmetry - Chess, Tic-Tac-Toe
Asymmetry – counter strike,battlefield,nintendo land,packman

Competition and Cooperation

- When players compete, they try to accomplish mutually exclusive goals
- When players cooperate, they work together to accomplish goals that are the same or similar
- Competition modes are ways to build cooperation and competition into games

Hiding the Rules

- Video games do not require written rules
  - The game enforces the rules
  - The player can't change the rules
- Provide adequate clues for players to overcome a challenge
  - Using trial and error to overcome a challenge frustrates many players

Setting the Pace

- The software determines the speed of the events in a game
- The player can't affect the speed of the game unless the software has to wait for player input
- The computer allows for modulation of the pace, so players can rest between periods of intense activity

Presenting a Game World

- To present a game world, video games can use
  - Pictures
  - Animation
  - Movies
  - Music
  - Dialog
  - Sound effects
  - Text and subtitles

Artificial Intelligence

- Today, artificial intelligence is used for
  - Strategy
  - Pathfinding
  - Simulating the behavior of people and creatures
- As artificial intelligence advances, games will add more uses for it
  - Natural language parsing
  - Natural language generation
  - Pattern recognition

Intelligent behaviour

Aesthetics

- All game elements should be high in quality and present a harmonious look and feel
  - The look includes the quality and appearance of the graphics, movie clips, animation, buttons, and fonts
  - The feel includes the music, dialog, user interface, and objectives
- Harmony is the feeling that all game elements are part of a coherent whole

Storytelling

- Most games incorporate some kind of story
- Video games can mix storylike and gamelike entertainment almost seamlessly
  - They can make player feel he is inside a story and affecting flow of events
  - This is one reason why video games are considered a new medium

Risks and Rewards

- Risk is created by uncertainty
- If the player overcomes the risk, a reward should be given
  - The size of the reward should match the size of the risk

Novelty and Learning

- Video games can offer more variety and content than any traditional game
- Novelty alone is not enough to sustain player interest, however
- People enjoy learning when it takes place in an enjoyable context or provides useful mastery
- Games should supply both enjoyable context and useful mastery

Creative and Expressive Play

- People love to select, design, and customize
- This activity can have a direct effect on gameplay
- As video games reach a wider audience, creative and expressive play become increasingly important

Immersion

- Immersed players lose track of the real world outside the game
- Immersion can be
  - Tactical—the sense of being "in the groove"
  - Strategic—observing, calculating, planning
  - Narrative—the feeling of being inside a story

Socializing

- Most traditional games are played with other people, making it a social activity
- Several methods allow people to play video games together
  - Multiplayer local
  - Networked play (multiplayer distributed)
  - LAN parties
  - Group play

Game design principles

The basic principles of game design include the following:

- The game design should be kept simple.
- Every game should be unique.
- There should be effective representation to bring out the real- life environment.
- It should include social factors, like it takes two to play a game.
- Playing the game should be fun.

Game design Process

The basic steps of the **Game design Process** :

Step 1 : Build the concept

1. Get an idea.
2. Build the game concept.
3. Create a goal.
4. Create an emotional experience for the player.

Step 2 : Creating the game specification

1. Write the design document.
2. Build the prototype.
3. Iterate, that is, choose lifecycle model.

A. Build the concept

1. Getting an idea

- Dreams and inspirations can create ideas.
- come up with a unique and unusual idea for a game.
- brainstorm sessions for creating an idea.
- Pen down ideas.
- evaluate ideas based on parameters such as originality, potential audience.

selected ideas are then subjected to synergy check and the concept is created.

2. Build the game concept

- A game is good if the player can win by doing unexpected things.
- surprise and delight factor almost make the gameplay, which encourages the player to employ strategies.
- The broad genres of games are

  action (use many buttons/keys),

  adventure (story is important),

  puzzle (analytical thinking),

  educational (learning by doing) and

  strategies (non-trivial choice making).

Games like movies can combine genres: for example, an adventure game can combine action and puzzle.

3. Create a goal

- Every game has a goal.
- All aspects of the game must focus the player towards achieving the goal.

All games have one or more of the following goals:

- To collect points (do not have a visible objective.
- To gain power.
- To reach the destination/target first.(racing game)
- To overcome obstacles.
- To discover something.
- To eliminate opponents.

Point-Scoring Vs. Racing and Conquest-type game

- The point-scoring games , rewards are given in the form of points - do not have a visible objective,
- In racing and conquest-type game, both involve visible objective. For instance.

- In racing, the player knows what is his/her position compared to other participants at any point of time.-Mario cart, crazy taxi, need for speed,
- In role-playing games (RPGs), the points earned can be spent on improving the attributes.- Dungeons & Dragons,World of Warcraft, Final Fantasy
- In strategy games, the gathered resources are used on units.-Packman,Age of Empire,Civilization,Poker,Chess,Go
- In adventure games; the collected items are used to solve something later. –Heavy rain,Myst(93),The walking dead(2012),broken age(2014),Fahrenheit(2005)Kings Quest I(83)The WITNESS(2016),Real Myst etc.

4. Create emotional experience for player

- If the emotional experience does not match the game goal, the player would not enjoy the game.
- The programmer needs to design the game in such a way that player get addicted to the game.
- For this purpose, cool graphics, fast actions, stimulating steps or complex strategies could be added to the game.
- The features of the games such as score, moves, and levels could be designed as follows;
- **Score:** In fighting, racings, sports or any competition games, Players expect more from a game experience than a mere score.
- **Moves:** Controlling virtual athletes' in sports and vehicles in racing requires a steady practiced hand players must master all those tricky moves to almost devastate their opponents.
- **Explore levels:** exploration can be encouraged by - hiding some of the game levels. As the player moves through the levels he/she finds ways to unlock or realize the hidden areas. A racing game can have additional cars as the players acquire trophies a particular number of times. Similarly, a FreeCell game gets tougher as the player wins the first few games.

B. Creating the game specification

Steps

- **Create Plan first**

➢ First create a plan answering following Questions-

1. What type (genre) of game it is?
2. What is the objective?
3. How will player achieve the objective?
4. What are the features that will constitute the gameplay?

a) Identify the players. b) Identify the genre and corresponding constraints.

c) Identify the universe and landscape. d)constraints and goals of the games.

- Determine the criteria for success. How does the player win? How does he/she lose?
- Determine the rules of interaction. Is it through controls or interaction with character or by putting the player in some environment.
- How are the constraints explained to the player? Is a story told to the player?
- What are the operational issues (in production/maintenance)?

Creating the game specification - Test the concept

- The concept is put down on paper,
- writing the concept down exposes the issues and the complex interfaces.
- This treatment forms the outline of the concept and the sketch of game design.
- This ensures the potential of game and helps in deciding whether to go ahead or give up.
- At this stage, the technical constraints are not seen.
- Only the game’s unique features are considered and other details are ignored.

Creating the game specification - Feature- based description

- Feature- based description of the game is easy to understand.
- Features are emergent from rules and most of the rules keep on changing as we detail the game.
- The reason for the changes is that several rules interact with each other.
- Feature is emergent if new categories must be used to describe it, and are not required to describe the underlying rules.

Creating the game specification - Feature- based description

There are three types of features:

- Vital / integral features - make the game work properly. For example, in football, the formation is selected first, and then the choices of the players are made accordingly.
- chrome features - Some features have no effect on the game played, but enhance the enjoyment of the game. These features help one draw the story and convey the look and feel of the game. For example, in football, the player chooses the stadium; or in a game like *Star*Craft, the interface is customized to reflect the alien species that are chosen for playing.
- Gameplay substitutes -Some features do not enhance the game in any way, but merely give an equivalent choice (the choice makes no difference in achieving the objective). For example, the player can play one on one with computer or choose a multiplayer mode in LAN.

Creating the game specification - Feature- based description

The purpose of the gameplay is to -

- explain the developers how the game works;
- provide the vision statement referred throughout development;
- identify features that are integral and that are chrome.

Creating the game specification - Interface design

- A good interface design makes the player feel that there are no restrictions on the control of the game.
- Interface should efficiently handle a large number of simultaneous tasks in a complicated strategy game.
- Use the list (with scroll bars, if necessary) for selection.
- Multiple section facility should also be available, if necessary.

Creating the game specification - Rules

- At initial stage for creating a game, the rules have to be guessed.
- As the game development evolves iteratively, more rules are added and these rules interact with the previous ones, thereby making changes to them.
- The rules must serve the features.
- Share damage through the formation.
- If one member is hurt, the damage would be split to all units in the formation.
- Size and type of formation would increase the toughness.

Creating the game specification -Level design

- Level design affects the core gameplay.
- Levels depend on the storyline and can enhance the inherent gameplay or detract from it.
- Issues related to level design have to be addressed in the game specification itself.
- At its simplest level, low interactivity is given, which is reactive in entertainment software, which involves watching more than playing .
- In the more story-oriented genres that evolve out of adventure games, More flexible narratives are given so that the viewer can delve into it with as much interactivity as he/she wants.
- Level should not be used to cover deficiencies in the gameplay.

Creating the game specification -Level design

- Choices related to that level only should be visible in that level.
- An accurate specification for each level is given to the level designers, who understand the rules.
- Best level design is not linear, it allows multiple approaches.
- A good game should allow the player to make strategies (long-term and tactics (short term)
- The primary goals of a game design are to create interest in the reader and to convey the design aspects without ambiguity.
- The interest of the reader should sustain from beginning to end of the document.
- Reader should have a clear understanding of the design and nuance of the game in the genre.

Creating the game specification - Design document

General Information about brief description of the game

- **Basic concept:** The basic concept describes the basic idea in one or two pages. The idea should be concise, informative and interesting.
- **Background story:** This is necessary for adventure kind of games and nor for puzzle kind of games.
- **Objective:** The objective of the game is described. The reader should have a clear understanding of the gameplay , therefore as much detail as possible is given in the design document.
- **Gameplay:** The way the game works from beginning to end is described. After loading the game, a title screen or an option screen has to be shown. The choice that have to be provided in the option screen and can any of the option be bypassed, etc have to be decided.

Creating the game specification - Design document

- The result of choosing a particular option should be known to the player.
- The method of providing the result can be through Help option in the menu.
- The player's assumption of the character (Hero) and his enemy/opponent (controlled by computer) have to be kept in mind.
- The computerized opponent is described by artificial intelligence. (AI)

Creating the game specification - Design document

The detailed description of the following are decided.

- Interfaces that are to be defined.
- The different perspectives to be planned
- The basic interactive structure in terms of sections and levels.
- The genre of gameplay, if it needs speed and whether it has to be continuous as in racing or turn-based RPG, etc, to be noted.
- The game has to be written for single or multiplayer PC-based game or LAN game.
- The difficulty levels that are required. The targeted are group for the game.
- The duration for the gameplay that any average player will take.

Creating the game specification - Game design- the other aspects

Characters: For all the characters in the game, mention their personality, capability and role/action in the game.

World: All different scenes where action takes place have to be described. Design documents of RPGs are primarily organized by location, characters, objects and the events that happen in that location. If locations are visited in any sequence or by travelling through any shortest path, then

the optimum order is also mentioned.

Controls: The controls that have to be provided in the user interface are described. For any action to take place, buttons can be provided on the controller in the user interface. The state of the game is conveyed to the player though the user interface. For example, the player should know his/her scores, the resources left or life gauge, etc. The navigation through the game has to be well described. The player should be aware of the alternate ways of moving through the game . Wherever applicable, on-screen text message can be given.

Creating the game specification - Game design- the other aspects

- **Graphics:** The choice in the graphics mode if required has to be stated as choice for the player, the corresponding style to be chosen. For example. In the game of tennis-hard court, clay court, grass, etc, climatic conditions/weather options should be there. Pictures of major scenes in the game can be presented to visualize the game . Character sketches can be done to show how they will look.
- **Sounds and music:** Where and when (specific action performed by the player/occurrence offend event) the sound effect is required in order to create an impact in the game. The general music and at the opening of any new level. different sounds can be played. The laughing track, the spectators' reactions, noise made by animals, ghosts army weapons, etc are added to give a real-time effect.

Creating the game specification - Build prototype

- Prototype are built first, in order to minimize the risk;
- get the client feedback in development process early;
- help to visualize the end product;
- find the feasibility

Typical Game Sections

- **Game Section -1**
- The environment is set up for the game and initialized.
- Option to play or exit is entered.
- User is requested to enter input.
- User input is stored and viewed.

Game section -2

- Game state is changed according to user's last input.
- AI is applied on player's action, Then, background animation, music, sound effect are performed.
- Finally, housekeeping is done.

Game Section-3

- Current animation frame is drawn to virtual first, and then displayed on the screen.
- Frame display rate is specified and locked.
- In the exit routine, resources are released.
- System settings are restored.
- Control is given back t

Life cycle model

- Rapid prototyping phase before specification is a very popular technique.
- Building of game in increments (repeated loop) through merge design, implementation and integration phases into a repeated block.
- Agile –extreme programming involves

  team-based and very collaborative approach;

  hardly any documentation;

  integration of testing into the development process (instead of testing everything at the end, testing is done at certain milestones).

Life cycle model continued …

Development process involves :

- building a prototype;
- playing the game for a while and getting the basics right;
- asking other people to look at the game;
- meticulously observing the player;
- constantly rewarding the player in the game;
- using new testers;
- taking the game to next level of complexity;

Life cycle model continued …

Good designs are both

- high level (architectural level) and
- low level (detail design).

Even when changes are made in the detail design, the high-level architectural design is intact and

so the final goal of the game remains unaffected. The “chrome” elements add to the look, and the feel of the game has to be coherent to give an aesthetic appeal.

Gameplay

Definition : The Strategies that are required to reach various end points are collectively termed as gameplay.

Rules interact with other rules of the environment to produce a result.

Implementing gameplay

A game is a set of a series of interesting non-trivial choices.

A decision has gameplay value when it has an upside and a downside, and the overall payoffs also depend on other factors.

Each decision affects the next. The decisions include choosing the correct option in the circumstances and looking out for better tactics to apply.

Gameplay

- There is a dominated and dominant strategy problem in gameplay which are defined as follows;

Implementing gameplay

- The option which is not worth using under any circumstance is called a dominated strategy. - waste of time
- The option which is so appropriate that is not worth using any other option is called a dominant strategy.
- a dominant option even worse is a better options . Hence, a near-dominant option is the only one that is useful. It is used in narrow circumstance and it is the one that the player will use most often.
- An example of near dominated option is given as follows;

Ask the following question ;

- What is the most common feature?
- What should I do to make the most of it?
- If this facility is used only once in the game. How much of the development time has to be allocated for it?
- How much special effects and efforts for building the interface should be spend?

Gameplay Strategies

- A game has to be interesting even if the player knows all the tricks/tactics.
- If a careless player over looks a near-dominated option then the opponents can exploit the situation.
- However, near-dominance will show which options will get used most, and logically for those options more effort in terms of graphics, effects, etc, are spent.
- The programmer should ensure to include interesting choices in the game.

Different kinds of options a gameplay involves are as follows.

- Option may or may not be taken depending upon certain factors.
- Option can have time constraints and is context-dependent.
- Option does not affect the game much whether taken or not.
- Option that is worth being taken
- Option that is never being taken.

Toolbox to assist with choices

- There is no particular method for making the choice that are left to the creativity of the player.
- Ensuring gameplay choices are non-trivial, make the game interesting.
- Strategic choices are made for long term and tactical choices for the current or short term.
- However, strategic choices will affect the range of tactical choices that are made later.
- Player who selects options according to a set plan will not do as well as the player who adapts to the circumstances.
- There should be enough scope for good judgment, and this gives a better payoff.
- The payoffs for choosing different investments in strategic plan may alter the decision that the player will take at different levels.

Toolbox to assist with choices

- Speed is a kind of versatility feature.
- A unit that normally moves fast need not be having other best factors too.
- The compensating factors could be that we could make the unit weak or we could make the unit expensive to buy.
- If the unit is made expensive, then the player may or may not buy it. Once he/she has made the decision, he/she has committed to it. Hence the choice whether to buy or not may be strategically interesting but it is tactically not.
- It does not oblige the player to use the unit in any interesting way. The player has to understand what are the compensating factors.
- If the expensive unit is more powerful and would produce an impact, he/she might still go for it. The choice is entirely the player's and it is a well-informed choice.

Impermanence

In some decision, the payoff leads are long lasting.

The advantage can be impermanent, because of the following factors;

- They can be destroyed, injured or stolen.
- They can last only for some times or can have limited use.
- They are of no use in certain circumstances

Impermanence can also be a kind of choice the player has to make. While doing so, he/she has to understand the cost and trade-offs.

- The cost does not mean only money and points gained;
- it also means the time, efforts, attention and alternate resource required at that point.

Gameplay allows you to interact and enrich your experience. This interaction can be done in following ways.

- By changing the setting or game itself.
- By directly controlling the character.

Gameplay specification

Definition :

The gameplay specification is a “vision document” that describes the final product and serves as a goal for all developers.

When the code strictly adheres to gameplay specifications, then the product is ready for release.

The designer notes explaining the reasons for all assumptions is required along with specification, since this will help others to verify/challenges the assumptions at the initial stage itself.

Responsibility of the designer

- Process of good creative design is to filter and clarify the designer's ideas, which he has taken from many sources, and bring them to a structure that will best suit the game.
- designer has to discuss or arrange for a brainstorming session with the team and accept inputs from them.
- This will lead to gathering of more ideas and also reinforce/verify the ideas stated by the designer in the first instance.
- Apart from the development team, the product in view has to be presented to the marketing team to solicit their guidance of the commercial aspects so that the features in the game take care of the expectations of the clients at least for the next two years.

Designers

Responsible for the look and feel of a game.

- subroles: scriptwriter, level designer, ...

Game Designer

What my mom thinks I do

What my friends think I do

What society thinks I do

What programmers think I do

What I think I do

What I actually do

Art and animation staff

Responsible for the visual look of the game.

Sound and music engineers

Produce the music and sound effects for the game.

- also integrate the soundtrack into the game programming

Programmers

Create the code for game engines.

- subroles - graphics programmer, AI programmer

Quality Assurance and testing

Make sure the game works and meets its specifications.

- should have broad of games to compare against
- not just finds bugs but provides information to fix them
- responsible for the delivery

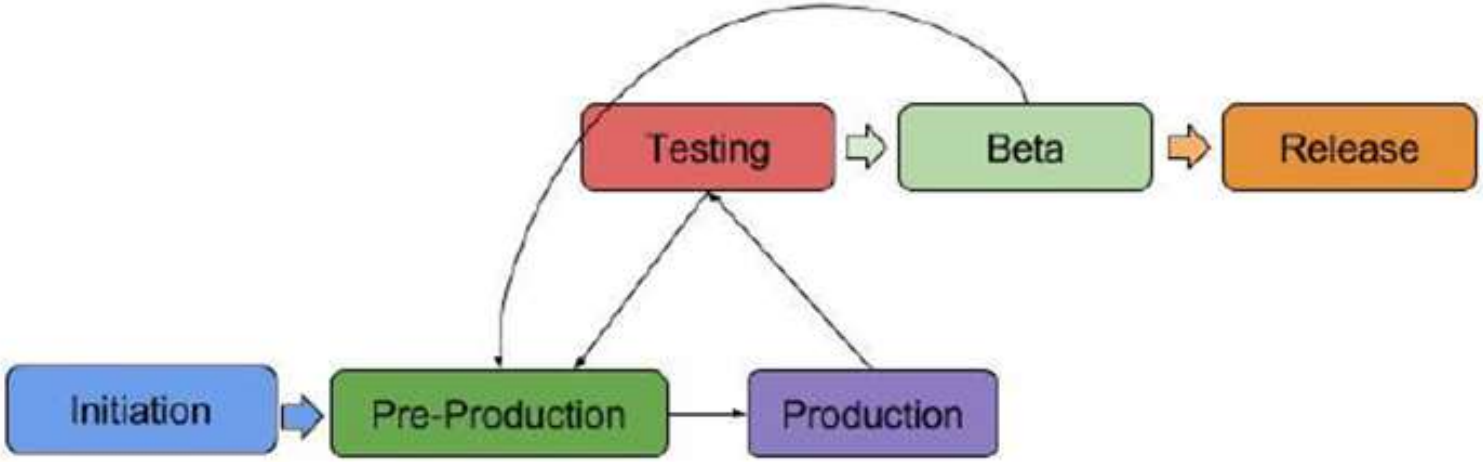
Managers

Oversee the entire project.

- make sure everything is running well and project is in schedule
- many subroles: art lead, programming lead
- someone has to think about the business

The Phases

The GDLC typically consists of six phases shown below:



Initiation phase

The developer decides what kind of game they will make.

1. Building a Story and Script

2. Feasibility Study

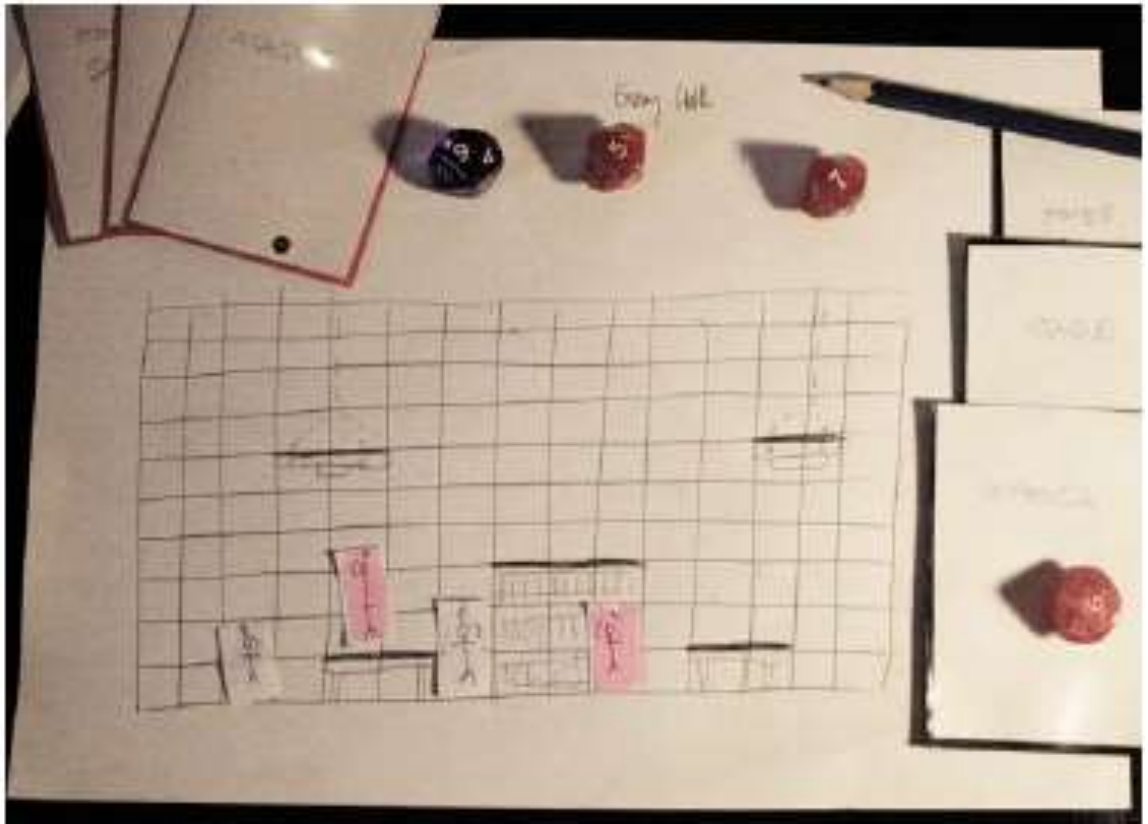
Pre-Production

Before the real production begins.

First pre-production:

- Game Design Document
- First prototype - shows gameplay (has to be fun)

Next cycles -> bug fixing and balancing



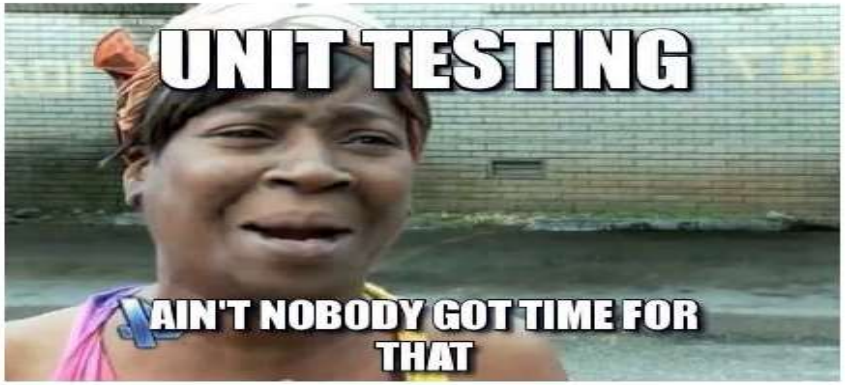
Production phase

The game assets and source code are made.
The result of production is the playable game in form of:

- **Formal Details prototype** - a playable game, has win-lose rules, co-relations between features, runs well.
- **Refinement prototype** - most mature prototype which only needs more polishing. Should be feature complete and almost ready to ship.

Testing phase

Evaluation of game features, value, concept and design.



Test report -> all the bugs, thing that should be omitted, included or changed.
The test result should decide whether to reiterate or advance to beta testing.

Some questions the testing should be answer:

1. Is the game still buggy?
2. Is it possible to get stuck in the game?
3. Is there any sign of exploits/glitch?
4. Is the game too easy/hard to beat?
5. Is there any feature missing?
6. Does the game run well on every platform?

Beta Phase

The beta test is testing cycle conducted by third party:

- publisher,
- potential buyer,
- game reviewer

Result should be a test report.

Decide whether the game is ready for shipping.

Release Phase

Work may seem to be done but it is not!

- Bugfixing, patching
- Aditions, special events
- Marketing
- Community management

Development phases in gameplay

various phases of development in gameplay are (8 phases)

Phase	Process	People	Outcome
Feasibility	Discussing the initial idea and feasibility	Designer, architect technology group	Project go ahead signal from client
Conceptualization	Detailing the design	Designer	Gameplay specification, designer's notes
Plan	planning the mini-specifications for each stage along with the objective to achieve	Designer and a software planner	Set of mini-specification document
Technical architecture	Preparing architectural components and tools that will be needed. This is then handed over to the project leader. Mini specifications are mapped to milestones in project plan	Architect, project leader, planner	Technical specifications project plan

Development phases in gameplay

various phases of development in gameplay are

Phase	Process	People	Outcome
Component Building	Constructing the components and tools required for the game. Considering interdependencies of components and completing the aspects of game architecture one by one	Development team	Components that are functionally complete
Integration	Project leader gets the components assembled by the development team, which are tested and reviewed by designer. This requires support from tools programmer	Project leader designer (only to review), tools programmer	Completed part of the game and tools

Development phases in gameplay

various phases of development in gameplay are

Phase	Process	People	Outcome
Complexity	Project leader in consultation with the level designer builds game levels	Project leader designer (one for each level)	Completed game for each level is ready along with tutorials/demos. Artwork and manuals are also finalized
Review	Quality assurance department does the review in parallel with leveling. This can also be outsourced	Quality assurance department	Bugs are fixed and feedback is sent to developers

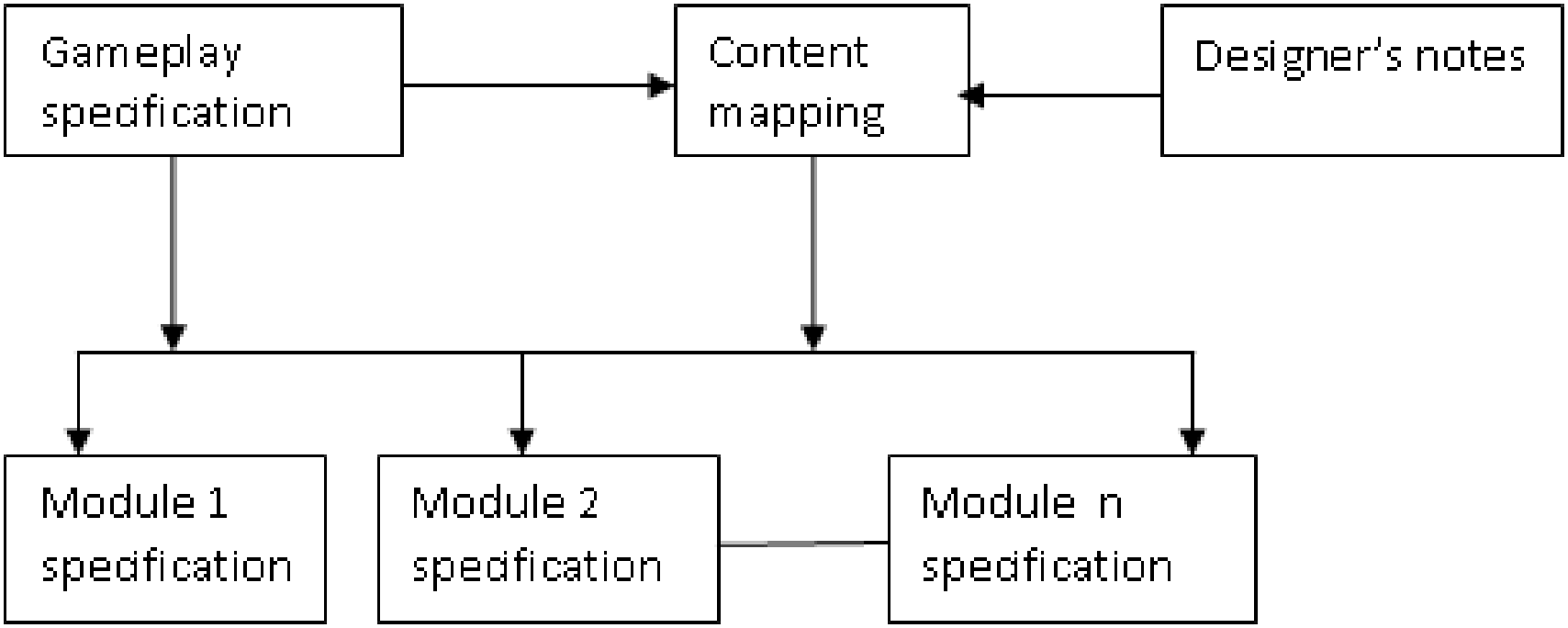
Development phases in gameplay

- Project leader and planner are involved in the release operation.
- Release does not mean the game is a complete finished product, but means that it is a usable product.
- It is considered finished only when all the features incorporated work perfectly and the required artwork, interface and other features are completed.
- Game design in evolutionary.
- The gameplay specification is an evolving document as changes have to be incorporated into the design document throughout the development process.
- Designer's notes reasons out the gameplay specifications.
- This is analogous to the comments in the code.
- If a change in design is made on a later stage in view of improving the performance, it may not hold good for a long time if one does not understand why it was not thought of in the first place.

Mapping design document with gameplay specification

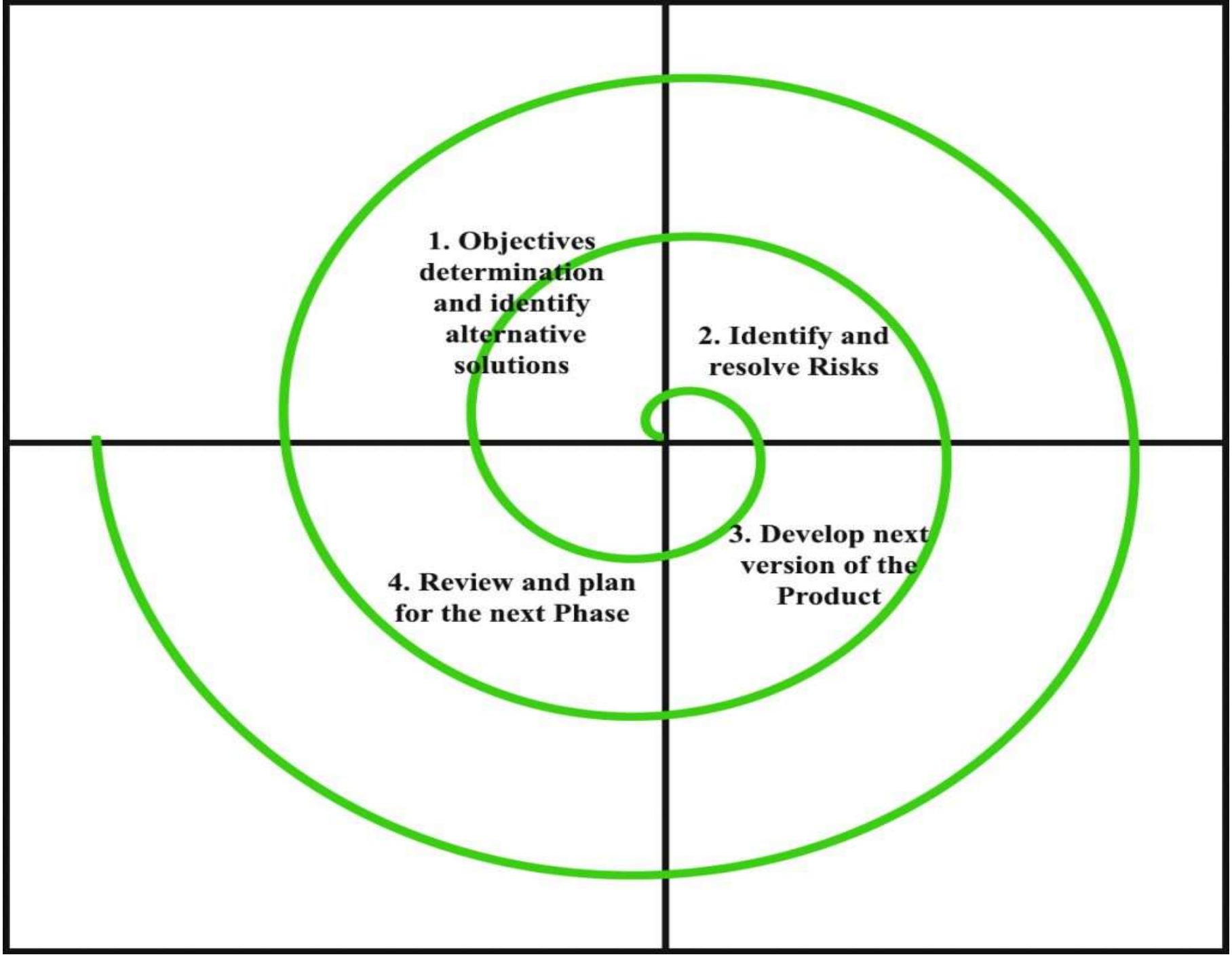
- It is mandatory for the programmers to refer to the design document while carrying out the tasks.
- First, it may have the answer to the queries they have in mind while development, and
- second it gives an opportunity to all developers to give their views on the design.
- The designer finally reviews their ideas and incorporates them if found suitable
- In reality, programmers rarely read these design documents though they are made available for them. Also, it is expensive to tag one member from the design group with every programmer. The solution to this problem can be as follows.
- A part of the design document that is relevant to the context of the code is made available to programmer instead of entire design document (Fig.)

Mapping design document with gameplay specification



Mapping design document with gameplay specification

- Content mapping uses the work breakdown structure of the Gameplay specification, maps it to the design document and extracts the relevant part of the document.
- Therefore, to every module the corresponding designer's notes is attached. The mapping can be in the from of hypertext links also, so that the developer can view the notes whenever he/she wants.
- Modularizing helps in making changes independently in each part of the design and test.
- Games are developed in an iterative manner, since the risk is high and so spiral process model is chosen most of the times.
- Most of the rules that the gameplay specifications document contains have to be tried out so as to find out how they work in practice.


1. Objectives determination and identify alternative solutions
2. Identify and resolve Risks
3. Develop next version of the Product
4. Review and plan for the next Phase

- If the prime key functions have to be tried out, then a prototype or test-bed model is used.
- In this model, the designer has the opportunity to improve the model as he proceeds with the specification.
- Any creative designer can produce an excellent game by using cannot track the task in the critical path, cannot estimate the time for completion, etc.
- Hence it is used only if the game is of an entirely new category which has not been tried before.

- In the iterative development, the gameplay specification is divided into a group of mini-specifications that defines a level.
- Each mini-specification (level) is developed and tested.

Each level has the following design items;

- Goal to design the features to be implement in that level.
- Philosophy to find what this level intends to do
- Expected results as to what the outcomes is at a particular stage.
- Alternatives or other ideas suggested during the brainstorming sessions.
- It is easy to develop the game architecture with these mini-specifications or level document.

2. Initial Design

- Game playing is enjoyed by Children as well as adult
- The way of looking at game is different.
- Children will appreciate the fun aspects of the game, but an adult too can enjoy entertainment software.
- Many adults play game like Age of Empires because they like ordering those little men around to build a city.
- What annoys them is when the enemy players storm in, destroy it and spoil the fun.
- I think adults like to have gameplay just to make what we are doing seem respectable and do not play just for fun.

- Many people who do not play games ; think that Playing game just kills the time, but it is not so.
- More awareness of entertainment software is needed to de-stress people.

Adv of game playing :

- Play can educate the mind,
- stimulate the imagination
- the creativity sharpens the wits,
- lightens the heart and enriches the soul.

In the future, we can see more individuals such as screenwriters, artists, novelists and board-game designers entering computer game design, which will revolutionize the world of entertainment software. What makes it truly entertaining is that many of these people will have no idea of the limitations of the technology (the designer need not worry about that, at least in the initial design phase.)They will demand the challenge; will deliver games astoundingly different from what we have already seen.

- Software architecture in general uses certain rules, heuristics and patterns;
- The problem is partitioned so that the system built can be easily maintained.
- The interfaces are created between these partitions.
- The overall structure is managed and the flow is made easy.
- Techniques are used to interface the system to its environment.
- Development and delivery approaches, techniques and tools can be properly planned and executed.

Architecture is important because of the following reasons:

- It controls complexity.
- It ensures best practices are followed.
- It takes care of consistency and uniformity.
- It increases predictability.
- It allows reuse.

Need for good architecture design

The advantage of a good architecture design is that,

- It describes and plan for the expected gameplay features,
- also gives us a rational framework within which we can make any required changes.
- If we do not have a design, we may find ourselves frantically gameplay the day before shipping which is never a good idea.

Architectural Design can be

An iterative or

tier-based design -more vital for any product that has to incorporate new or untried futures.

If we have to stick to a **deadline** and positively have to ship by a certain date, the **tier system** will let us fit the design to that date. If development is slipping, we can trim some features (or even drop them entirely), but the modularity inherent in the design will allow us to do so while still keeping the project under control.

- The architectural design in gaming evolves slowly and care has to be taken in keeping the documentation up to date.
- Game design has to satisfy many stakeholders, such as high-level management, marketing department and peers of the game designer.
- To ensure it is worth developing , the design has to go through a number of iterations and refinements throughout the development process.
- The game design had to be converted to technical specification in a way that the implementer can understand and code.
- The technical design acts as a roadmap and also increases projects visibility to generate an overall idea of how to work with project plan.
- The order will realize where his/her module will fit in the overall picture (class diagram).

Game and hardware abstraction

- The architecture design has to deal with both
- Game abstraction and
- Hardware abstraction interface.

- Abstraction emphasizes on the idea properties rather than how it works.
- It greatly reduces the complexity in large programs by hiding the irrelevant details.

Game abstraction

Game abstraction

- It is the way by which the game objects interact.
- Many gaming software objects are designed- communicated via APIs so that the logic in many of the gaming software modules may be designed independently of each other
- Modular gaming software architecture allows a **game-flow software module developed for similar group of games.**
- Game presentation software modules may be different to change the look and feel of the game.

Adv . Of Game abstraction approach

- simplifies the design
- ensures easy maintenance of the gaming software.
- software factory methodology provides the reusable objects for common functionalities.

- **Hardware abstraction**

Hardware abstraction

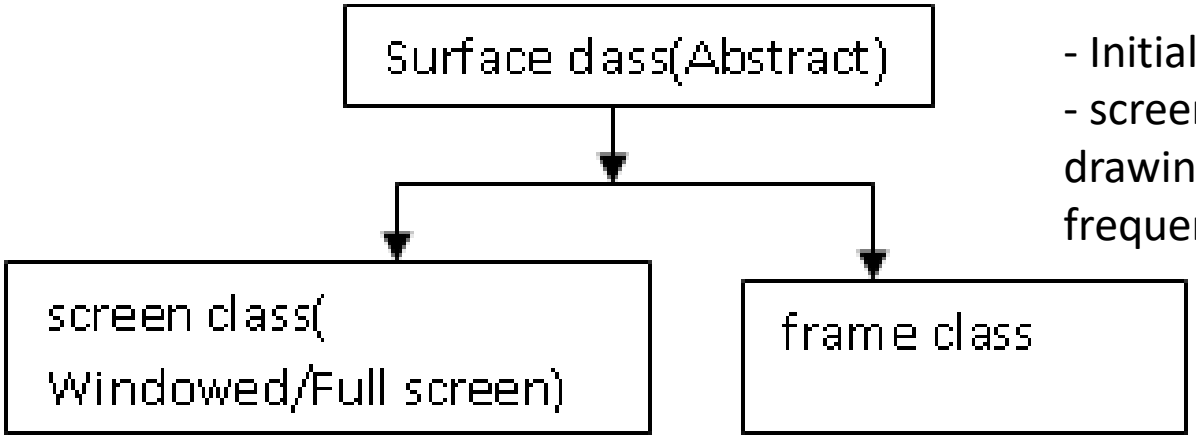
- Deals with the platform on which the software will be running.
- For every platform of different hardware configuration with different graphics card, sound card etc, and its codes have to be written and linked through the common interface.
- Hardware abstraction interface is not difficult since they already exist.

Graphics hardware abstraction

- Direct X - set of APIs that will call the appropriate driver in the hardware and also handle all the multimedia tasks.
- A graphical surface in DirectX can be created in either system (local) memory or video memory.
- surface initialization member takes parameters that indicate the height, width, bit depth of the surface and where it has to be located in the memory.
- The private members of the surface object provide error reporting, restore and release methods.

Graphics hardware abstraction

- Direct X - set of APIs that will call the appropriate driver in the hardware and also handle all the multimedia tasks.
- A graphical surface in DirectX can be created in either system (local) memory or video memory.
- surface initialization member takes parameters that indicate the height, width, bit depth of the surface and where it has to be located in the memory.
- The private members of the surface object provide error reporting, re



- Initialization code- called once
- screen-flipping and frame-drawing members are called frequently

Graphics hardware abstraction

- OpenGL which provides extensive cross-platform graphics library. For example, screen-handling library can support a common range of resolutions, double or triple buffering, bitmapped and True Type fonts, and sprites.
- graphics library translates the programmer's commands into device-specific commands. These commands specific are used to draw the graphical elements and object. Also, these commands are different for different devices, for example, commands for a plotter are different from that of a console.

Sound hardware abstraction

- sound card object is a single object accessed through an initialization function.
- The member functions of the sound card object are:
- playing, pausing, and stopping ;
- changing volume and frequency;
- specifying the position of the sound in three-dimensional space.
- The parameters can be enabled through the initialization function.
- No duplicate sound is made, and any new objects created refer to original data.
-

COTS- Components Off The Shelf

- All the basic functionalities- the sound system. graphics engine, music, saving of game. AI, artwork,etc. are available as **components off-the shelf** (COTS)
- COTS reduces the burden on programmers.
- Allow the game developers to concentrate more on gameplay.

Architectural design

- The objective of a good architectural design is to choose the best way of doing things from among the alternative designs.
- For proper game development, the game architecture should be understood properly.
- Without a proper architecture, the game is likely to result in high coupling.
- This will end up in procedural entanglement where the game loop does a million and one things.
- It is better to think about how things should work as a whole before starting the coding straight away.

Characteristics of good Architectural design

The following point applies to larger games where there are programmers, artists and designers all working on a game.

- **Simple feature:** A simple feature with a good tool that allows the developer to experiment and to iterate rapidly than out of a complex feature with no editor support.
- **Iteration for good results:** It should be easy for the people building the game to test the iterations while designing a feature.
- **Simple runtime code:** If major chunk of code is only required when editing a level, then it should be kept out of the game code. This mean a smaller code footprint, easier porting and less testing.

Characteristics of good Architectural design

- **Data –driven code:** Anything that might need to be adjusted should be a parameter that comes from data, not hard coded. Constants should be avoided and designers should not be allowed to set up complex gameplay conditions in scripts or using a visual scripting language. Designing your code around data also helps when trying to multi-thread systems for taking advantage of the console hardware.
- **Hard to debug data-driven engines:** Though using data-driven engines has its advantages, one major disadvantage of using them is that when a problem arises, these engines are hard to debug. To overcome any such problem that might inevitably arise, some extra functionality needs to be added. To have true data-driven architecture, a game engine must contain reusability, structured scripting language, scripting/editing tools, and essentially a friendly architecture.

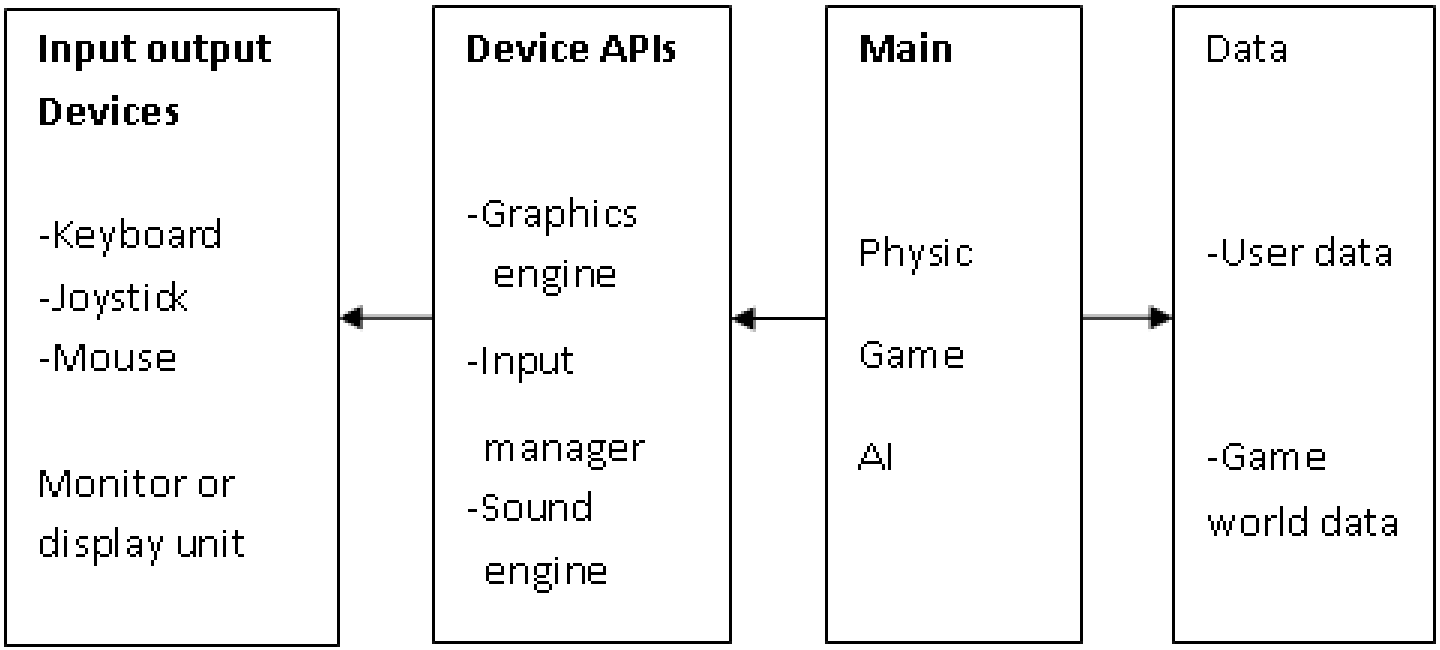
Characteristics of good Architectural design

- Scripting language is created to control AI.
- Script is required to be translated to C++ or bytecode.
- Vocabulary is required for interacting with the game engine.
- To connect scripting vocabulary to game engine internals, a “glue layer” is added. This allows pluggable AI modules, even after the release of the game.

Characteristics of good Architectural design

- **Game engines and graphic system:** There are number of free game engine or graphic systems that can be used to begin with designing the game.
- Writing a 3D game engine or similar engine takes a long time and requires many people to get it done.
- Using a different game engine and just coding those parts is much simpler.

Tiers of game architecture



1. **Input-output devices:** A large amount of data from the input and output devices are generated - read screaming.
2. **Device APIs:** The data stream generated needs to be processed by one or more engines, such as the graphics engine, input manager and /or the sound engine.

3. **Main -** The CPU's processing time is mostly spent on marshalling (assemble and arrange in order) the processed data from the game engines and performing general purpose logic. Smaller portions of the frame are used, as they are mostly devoted to physics, animation or graphics.

   The AI and the gameplay code pose a great challenge in the form of numerous exceptions, tunable parameters, indirections, etc. These are responsible to incorporate distinguishing features for each game.

4. **Data :** The data that has been processed by various engines, marshaled by the CPU and incorporated with distinguishing features by the API and gameplay code, has to be streamed out to the video display console for creating colorful graphics and giving a visual appeal. This process is called **write streaming or rendering**. In other words, it also refers to the transfer of data from the CPU to the GPU, to the frame buffer, which contains a small fraction of the frame.

The other reusable components are as follows:

- Setup programs (if required) for loading, saving and restoring of games and data.
- Menu system that helps in configuring the game settings, such as sound or music volume, and for redefining control and screen resolutions.
- Frameworks like DirectX that support the internal structures required to implement these features .

Use of good middleware

- Middleware is written to support the most common APIs used for physics, audio, animation, facial animation, network transport and various other systems.
- Middleware vendors can keep the cost lower than the developers themselves making the same code.
- Vendors can afford to keep larger, more experienced teams working on a given piece of functionality as compared to independent game developers because the vendors can sell the piece of code to many developers. It is wise to use a middleware unless these are specific needs that differ from the normal conditions.
- Middleware offers structure. It draws a line between the things that you have to worry about and the things that you do not need to worry about. When we use a middleware, we need to accept that technology's inflexibility, and modify our game design to suit it.

Tokenization

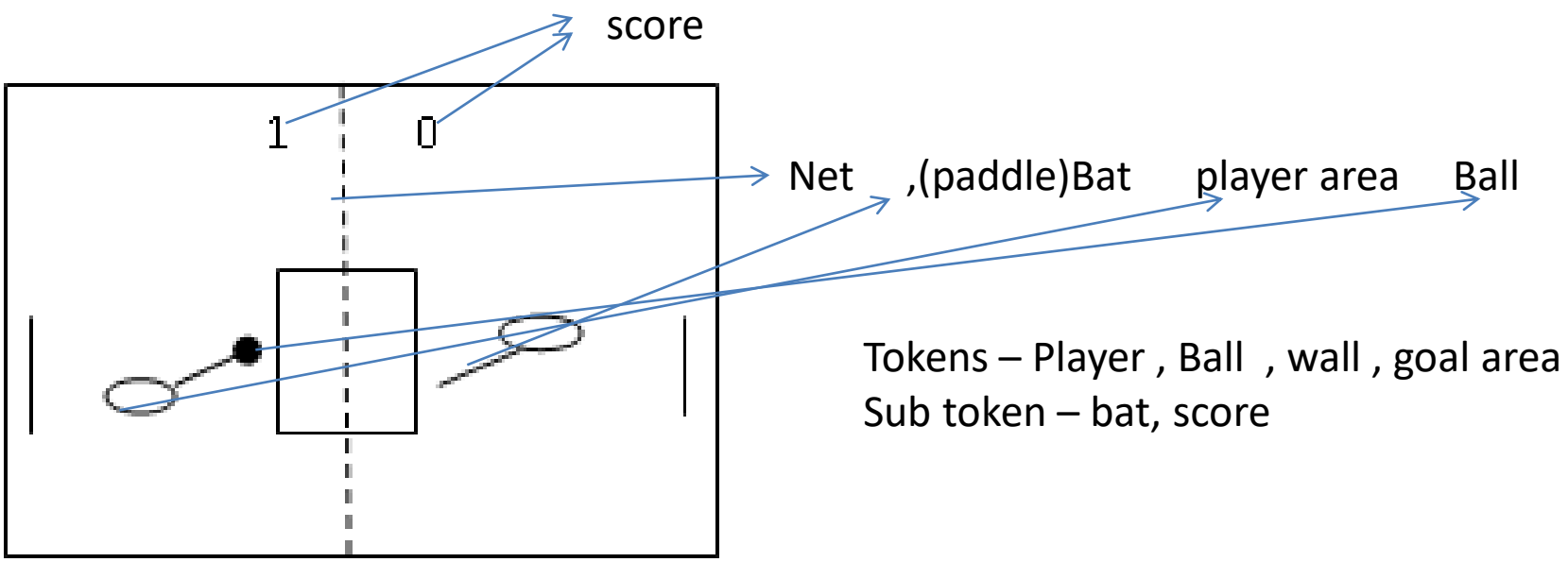
- Tokens are discrete elements that are directly or indirectly controlled and manipulated by the player.
- All the game scan is described interim of players and tokens.
- The tokens are conceptual objects which will eventually be translated into programming language objects.
- These might include a player's avatar, weapons, cards, counters metal figures, poker chips and letters of the alphabet (in a word game.)

Tokenization

- Tokens can be arranged in a form of hierarchy with the game world (play area) at its top.
- Every token has to be part of the game world token.
- For example, the player's avatar token acts as a user interface between the player and the game.

Tokenization of Pong

- Pongs is two-dimensional graphics video game
- Released by a firm Atari Incorporation in 1972.
- It is actually a simulated table-tennis game.
- Represented in 2D graphics.
- Player scores points if opponent fail to return a ball



- Tokens do not interact with each other on their own.
- An occurrence of an event causes the interaction.
- There are two types of event in Pong.
- Collision event and the goal event.
- The collision event is generated when two tokens collide.
-Each of these tokens gets a message that it has collided and the type of token it has coiled with.
-A token-interaction matrix is constructed, which exhibits all possible collision
-**Symmetric interaction:** If interaction is same in both the ways, for example, bat-ball and ball- bat, then the interaction is symmetric
-**Asymmetric integration:** This interaction depends on the direction. For example, the goal increments the score by I, whereas the score increment need not be caused by a goal.

	Bat	Ball	Wall	Goal	Score
Bat	x	-	-	-	-
Ball	Defection	x	-	-	-
Wall	Stop	Defection	x	-	-
Goal	x	Triggers goal event	x	x	x
Score	x	x	x	Goal score	x

x - No Interaction - means Symmetric Interaction

The uses of the game world token

- It acts as an intermediary between different tokens and also responds to certain other events. For example, it resets the game when the goal event occurs (and the score is incremented). If instead of game-world token, there has to be an inter-token communication, then each token has to have an intimate knowledge about the other tokens with which it may interact.
- Also, here the game-world token acts as an event handler. This leads to **design extensibility**, that is, when a new token needs to be added by the designer, it can be added easily.
- It can also do filtering events based on location.

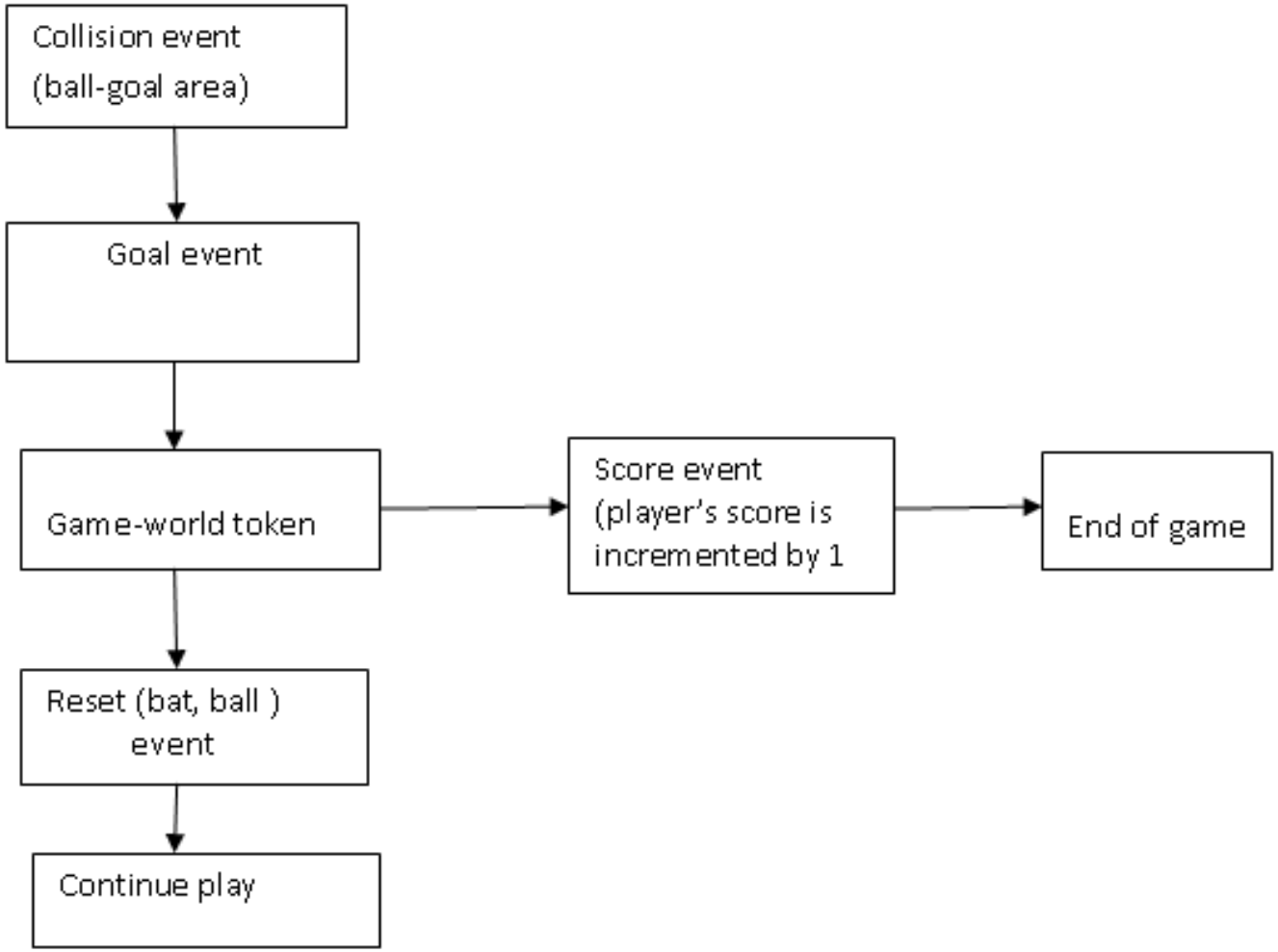


Fig. Sequence of events that occur when a goal is scored

- The token-interaction matrix also comes in handy when deciding the object-oriented architecture .
- We need to decide which of the features would be considered in the iteration.
- Suppose we plan to add graphics, then **CGameWorld class** will act as a container.
- Then we need to provide interface inheritance from multiple objects such as IDrawObject (to present it on the screen), an IScriptObject (to position it) and ILogObject (to debug the output) to a common class **CCommon** Token.

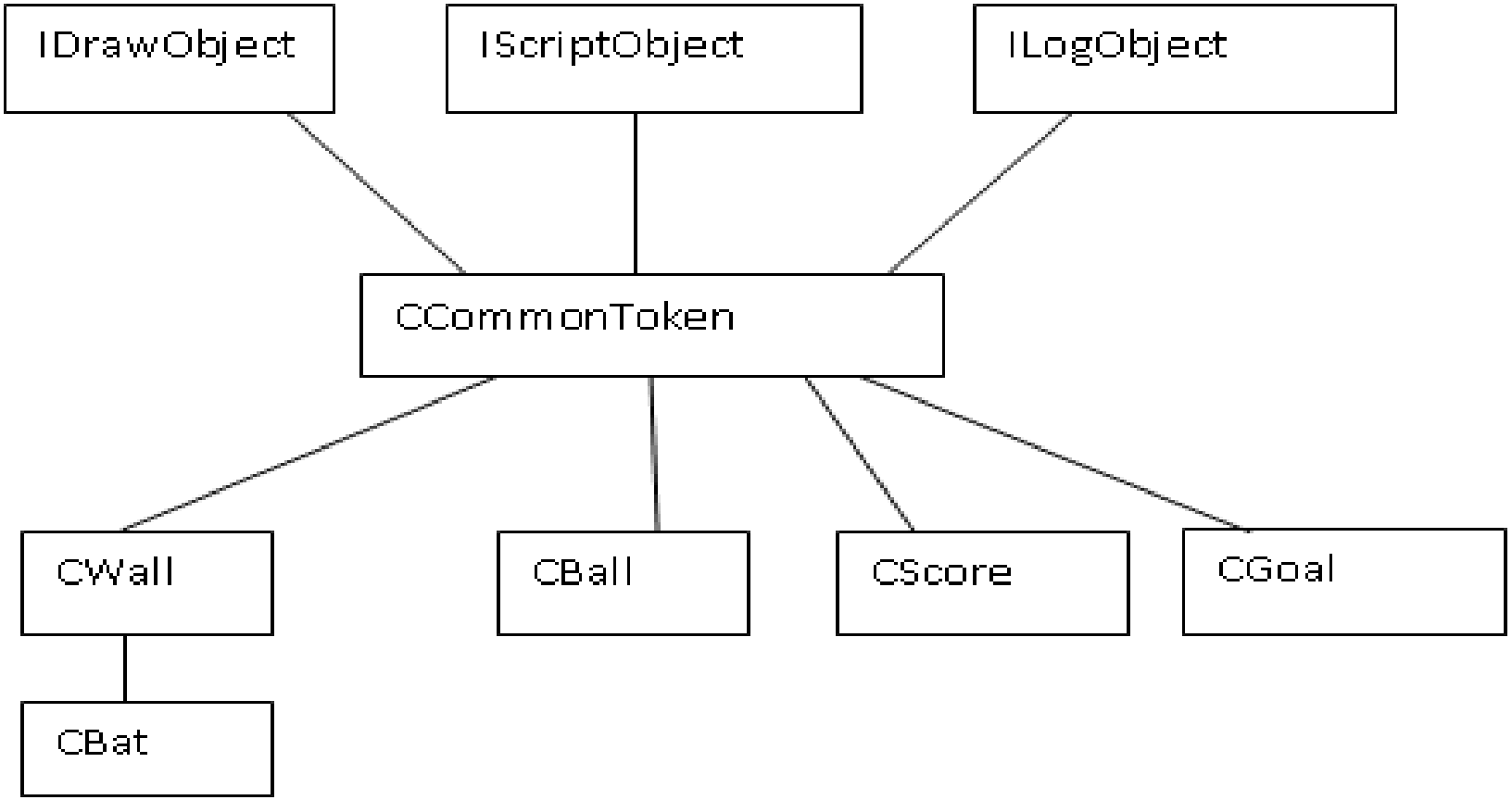


Fig. : Class hierarchy for the tokens of Pong Game

All the other tokens are derived from the common class CCommon Token, common class is derived from the interface Classes. The tokens that are derived from CCommonToken are direct token-to-object mappings such as CBall, CScore, CGoal and CWall.

The CBat is considered as a mobile piece of wall, so it is derived from the CWall class

State transition and properties

- Token will not instantaneously change from one state to another
- It takes some time and follows the transformation process.
- i.e. it takes a finite amount of time to reach a transitional state.
- Mostly token responds to only one event at a time
- Handling collision event: to sum the collision vectors to produce a net result
- For above result, one has to keep a track of all collision members
- Change in a design and add more tokens
- This shows architecture is not well structured

- The architecture should be flexible enough to add or remove tokens.
- Maintainability and expandability are the main goals of design
- Adding a new token should create minimal changes to the code (hard architecture) and majority of changes have to be added in the data the drives the game (soft architecture)
- architecture encompasses the structure (component and their interactions) and the data flow
- Analysis various problems system architect develop architectural model
- Architectural design breaks the game into subsystems which are built in stages

- "Hard" and "soft" architecture
  - Hard-> horizontal solution
    - It is a architectural model in which subsystems interface with the computer hardware and the player (i/p & o/p interface)
    - It uses standard API's such as DirectX, SDL, OpenGL and so remains almost static
  - Soft-> vertical solution
    - It actually makes a game
    - It is domain specific and is generally not reusable between game projects
    - As we progress a game, the code becomes less generic and more specific
    - Eg. Level-loading games , it was required to virtually load the soft architecture of the entire level and configure it at runtime

Best practices and points to remember

- The enterprise applications do not prepare one for programming game
- Because games are such specialized applications, the it is nearly impossible to give any one method for all
- One model is best for one and useless for other, it is very difficult to suggest a common model that works perfectly well for all
- Also theirs is no such things as an "industry standard" defined for the texture, mesh, level, sound or animation formats
- Each programmer just put his/her idea or uses whatever is convenient for the platform.
- Following things must be kept in mind by programmer while development

1. Use simple formats that you are familiar with. Do not worry about how they would work in larger projects. Use free fonts.
2. Do not worry about plug-ins, editors etc
3. When the first working model is ready, consider other elements, and then iterate