User Interfaces

What is the User Interface?
“What works is better than what looks good. The looks good can change, but what works, works”
UI lies between the player and the internals of the game. It translates player’s input into actions in the game world, passing on those actions to the core mechanics, and it presents the internal data that the player needs in each situation in visible and audible forms.
The outputs are referred as visual elements and audio elements and the inputs are referred as the control elements.

Player-Centric Interface Design

• How is the Game UI design different from general UI design?
  - Computer programs are tools, so the interfaces allow users to enter and create data, control processes, and see results.
  - Video games exists to entertain, UI doesn’t tell the player everything and it doesn’t give complete control over game, creating an experience for the player that feels like gameplay and storytelling

• Innovation
  - Don’t innovate unnecessarily when designing a new interface (button assignments on controllers and keyboards)
  - If a standard exists, use it (or as much of it as works with your game)
  - If you force the player to learn an unfamiliar UI when a perfectly good one already exists, you will frustrate him and he will dislike the game no matter what other good qualities it has
  - If you use a new interface, build a tutorial level and playtest if with novice and experience players. If it doesn’t have substantial improvement, go back to what works
  - Allow player to customize the interface, especially for advanced players

• General Principles
  - Be consistent: applies to aesthetic and functional issues. If you offer the same action in different gameplay modes, assign that action to the same controller button or menu item. The names that appear in indicators, menus and the main view should be identical in each location.
  - Give good feedback: when the player presses any screen button, the game should produce an audible response even if the button is inactive at the time.
  - Remember that the player is the one in control: Don’t seize the control of the avatar and make him do something the player may not want.
  - Limit the number of steps required to take an action: set a maximum of three controller-button presses to initiate any special move unless you need combo moves for a fighting game.
  - Permit easy reversal of actions: allow player to undo an action, unless it would affect the game balance adversely. Ex: puzzle games with cards or tiles
  - Minimize physical stress: assign common and rapid actions to the most easily accessible controller buttons. Not only do you reduce the chance of injuring your player, but you allow him to play longer and enjoy it more.
- Don’t strain the player’s short-term memory: don’t require the player to remember too many things at once, prove a way for him to look up information that he needs.

- Group related screen-based controls and feedback mechanisms on the screen: that way, the player can take in the information he needs in a single glance rather than having to look all over the screen to gather the information to make a decision.

- Provide shortcuts for experienced players

**What the player needs to know**

- Where am I?
  - Provide a view of the game world, if she can’t see the whole world at one time, also give her a map or a mini-map that enables her to orient herself with respect to parts of the world that she can’t currently see. Audio feedback from the world: ambient sounds that say something about the environment.

- What am I actually doing right now?
  - To tell the player what she’s doing, show her avatar, party units, or whatever she’s controlling in the game world, so she can see it moving, fighting, resting, and so on.

- What challenges am I facing?
  - Display the game’s challenges, puzzles, combat or whatever they may be.

- Did my action succeed or fail?
  - Show animations and indicators that display the consequences of actions. Accompany these consequences with a suitable audio feedback for both success and failure.

- How did I do?
  - Give the player emotional rewards for success and disincentives for failure through text messages, animations, and sounds

- Am I making progress?
  - Show indicators for the score, percentage of a task completed, or the fact that a player passed a checkpoint.

- Do I have what I need to play successfully?
  - The player must know what resources she can control and expend. Display indicators for each: ammunition, money, energy, and so on.

- Am I in danger of losing the game?
  - Show indicators for health points, power, time remaining in a timed challenge, or any other resource that must not be allowed to reach zero.

- What should I do next?
  - Players need guidance about what to do. You need to make sure that they always have an idea of what the next action could or should be.

**What the player wants to do**

- Move
  - Most video games include travel through the game world as a basic player action.

- Look around
  - In most games, the player cannot see the whole game world at one time, he needs a way of adjusting his view of the world.

- Interact physically with non-player characters
  - This usually means attacking, giving items from inventory, carrying or healing them, and many other kinds of interactions.

- Pick portable objects up and put them down
If the game includes portable objects, implement a mechanism for picking them up and putting them down. This can mean anything from picking up a chess piece or a full-blown inventory system. Be sure to include checks to prevent items from being put down in inappropriate places.

- Manipulate fixed objects
  This includes objects that can be manipulated but not picked up. For avatar-based games, design a mechanism that works whenever the avatar is close enough to the object to interact with it. In other models, let the player interact by clicking them. Limit the number of states: light is on or off, door is open or closed.

- Construct and demolish objects
  The game needs suitable control mechanisms for choosing something to build or materials to build with, selecting a place to build, and demolishing or disassembling already-built objects. It also needs to provide feedback.

- Conduct negotiations and financial transactions and set numeric values
  If you need to let the player manipulate raw numbers, try to find a way to make it fit into your game’s cultural style, so that it doesn’t harm the player’s fantasy.

- Give orders to units or characters
  Players need to give orders to units or characters. This requires two or three steps: designating the unit to receive the order, giving the order, and optionally giving the object of the order or target.

- Conduct conversations with non-player characters
  Video games almost always implement dialog with NPCs as scripted conversations conducted through a series of menus on the screen.

- Customize a character or vehicle
  Provide a suitable gameplay mode or shell menu. The player may want to customize visible attributes (hair, clothing, body type..) or invisible ones.

- Talk to friends in networked multiplayer games:
  Multiplayer online games should give the player opportunities to socialize. Chat systems, online bulletin board or forums.

- Pause the game:
  Single-player games must allow the player to pause the action (expect arcade games)

- Set game options:
  Set options like game difficulty level, customize the control assignments or adjust other features like the behavior of the camera. Use shell menus to do this.

- Save game:
  You must give the player a way to stop the game and continue from the same point when the player next starts up the game software.

- End the game: Don’t forget to include a way to quit!

The Design Process

The game design process takes place in three stages: concept, elaboration, and tuning. Designing the UI takes places early during the elaboration stage. There’s no point in designing it any earlier because you would have to change it.

- Define the gameplay modes first
  A gameplay mode consists of a camera model, an interaction model, and the gameplay (challenges and actions) available.
Design primary gameplay mode and then you can begin to create the details of the user interface for that mode.

- **Choosing a screen layout**
  Once you have a clear understanding of what the player does in the mode and you have chosen an interaction model and a camera model, you must choose the general screen layout and the visual elements that it will include.
  Main view of the game world should be the largest visual element on the screen but you need to find a balance between the screen space for the main view and the feedback elements and on-screen controls.

- **Telling the player what he needs to know**
  Depending on the gameplay chosen, use the general list from “What the player needs to know” earlier in this chapter. Keep in mind the “General Principles” listed earlier in this chapter.

- **Letting the player do what she wants**
  Devise an appropriate control mechanism to initiate every action the player can take that affects the game, refer to the list provided in “What the Player Wants to Do”. Based on the interaction model chosen, map the input devices to the player’s actions. Study other games in the same genre to see how they use on-screen buttons and menus or the physical button, joysticks and other gadgets on control devices.

- **Shell menus:**
  Allow the player to start, configure, and otherwise manage the operation of the game before and after play. It is the first thing your player will see when he starts up the game, so you don’t want to make a bad impression before the player even gets into the game world.

### Managing Complexity

- **Simplify the game:** if your game is too complex, make it simpler:
  - **Abstraction:**
    Remove a more accurate and detailed version of an aspect or function of a game and replace it with a less accurate and detailed version or no version at all. This makes the game less realistic, but easier to play. Example: driving games don’t simulate fuel consumption.
  - **Automation:**
    Let the computer handle a process, instead of being controlled by the player. Racing games usually automate the process of shifting gears so it just happens by itself; the player doesn’t have to think about it. You can let the player choose between automated or manual, and give a reward to the player for choosing the more complex task.

- **Depth vs Breadth**
  - **Broad interfaces:** provide a large number of options simultaneously. It allows the player to search the whole interface by looking for what he wants, but finding one item takes time. Once the player learns where to find the buttons or dials, he can usually find them again quickly.
  - **Deep interfaces:** offer only a few options at a time and require the player to make several selections in a row to get to the one he wants. They offer all their choices through a hierarchical series of menus or dialog boxes. Menus must be named and organized coherently to guide the player. It takes no training.
It’s a good idea to offer both: deep from the new players, broad for the experienced ones. If you can only offer one, try to make the breadth and depth roughly equal; but avoid making anything more than three or four levels deep if you can help it.

• Context-sensitive interfaces
  It shows the player only the options that she may actually use at the moment. Menu options that make no sense in the current context have a different color, or simply do not display. Graphic adventures, role-playing games and other mouse-controlled games often use a context-sensitive pointer that changes form when pointed at an object with which it can interact.

• Avoiding obscurity (when a player can’t actually tell what the button and menus do)
  - Artistic overenthusiasm: artists sometimes produce UI elements that, while attractive, convey no meaning.
  - The pressure to reduce UI screen usage: using an icon instead of a text label on a screen button saves space but icons can’t convey complicated messages as well as text can.
  - Developer familiarity with the material: always test your UI on someone unfamiliar with your game, see whether your test subjects can figure out for themselves how things work.

**Interaction Models**

Relationship between the player’s input via the input devices and the resulting actions in the game world. You create the game’s interaction model by deciding how the player’s controller-button presses and other real-world actions will be interpreted as game world activities by the core mechanics.

• Avatar-based: the player acts upon the world through the avatar and generally can influence only the region of the game world that the avatar currently inhabits. To implement this mode, many of the button-assignment decisions will center on navigation.

• Multipresent(omnipresent): player can act upon several different parts of the game world at a time. You need a camera model that permits him to see the various areas that he can change, usually an aerial perspective.

• Party-based Model: most commonly found in role-playing games, small groups of characters generally remain together. This model usually needs a point-and-click navigation and an aerial perspective.

• Contestant Model: the player answers questions and makes decisions, as if a contestant in a TV game show.

• Desktop Model: it mimics a computer desktop and is ordinarily found only in games that represent some kind of office activity, such as business simulations.

If you decide to create a different model, you may need to design more detailed tutorial levels to teach your player the controls.

**Camera Models**

To define a camera model, some design decisions have to be made about how you want the player to view the game world, what the camera focuses on, and how the camera behaves. Certain camera models work best with particular interaction models.

• 3D versus 2D:
  - Most large standalone games running on PCs or home game consoles employ 3D
  - Small games and those played within a web browser often still use 2D graphics.
  - Use 3D provided that you have the tools, the skills, and the time to do it well, good-looking 2D graphics are always preferable to bad-looking 3D graphics. Bad graphics show up from the first moment.

Context-sensitive interfaces
- It shows the player only the options that she may actually use at the moment. Menu options that make no sense in the current context have a different color, or simply do not display. Graphic adventures, role-playing games and other mouse-controlled games often use a context-sensitive pointer that changes form when pointed at an object with which it can interact.

Avoiding obscurity (when a player can’t actually tell what the button and menus do)
- Artistic overenthusiasm: artists sometimes produce UI elements that, while attractive, convey no meaning.
- The pressure to reduce UI screen usage: using an icon instead of a text label on a screen button saves space but icons can’t convey complicated messages as well as text can.
- Developer familiarity with the material: always test your UI on someone unfamiliar with your game, see whether your test subjects can figure out for themselves how things work.

Interaction Models
Relationship between the player’s input via the input devices and the resulting actions in the game world. You create the game’s interaction model by deciding how the player’s controller-button presses and other real-world actions will be interpreted as game world activities by the core mechanics.

• Avatar-based: the player acts upon the world through the avatar and generally can influence only the region of the game world that the avatar currently inhabits. To implement this mode, many of the button-assignment decisions will center on navigation.

• Multipresent(omnipresent): player can act upon several different parts of the game world at a time. You need a camera model that permits him to see the various areas that he can change, usually an aerial perspective.

• Party-based Model: most commonly found in role-playing games, small groups of characters generally remain together. This model usually needs a point-and-click navigation and an aerial perspective.

• Contestant Model: the player answers questions and makes decisions, as if a contestant in a TV game show.

• Desktop Model: it mimics a computer desktop and is ordinarily found only in games that represent some kind of office activity, such as business simulations.

If you decide to create a different model, you may need to design more detailed tutorial levels to teach your player the controls.

Camera Models
To define a camera model, some design decisions have to be made about how you want the player to view the game world, what the camera focuses on, and how the camera behaves. Certain camera models work best with particular interaction models.

• 3D versus 2D:
  - Most large standalone games running on PCs or home game consoles employ 3D
  - Small games and those played within a web browser often still use 2D graphics.
  - Use 3D provided that you have the tools, the skills, and the time to do it well, good-looking 2D graphics are always preferable to bad-looking 3D graphics. Bad graphics show up from the first moment.
"If you cannot do it well, don’t do it at all"

- First-Person Perspective: used in avatar-based gameplay modes, the camera takes the position of the avatar’s own eyes, and is fixed with respect to the avatar. Player can’t see the avatar’s body, only the handheld weapons, or the avatar’s hands. Removes any need to adjust the camera, to look around, the player simply moves the avatar. Works well to display the point of view of the driver of a vehicle because it shows the terrain ahead as well as the vehicle’s instrument panel.
  - Advantages:
    a. Cuts development costs significantly because the artists don’t have to develop a large number of animations of the avatar.
    b. You won’t need to design AI to control the camera. The camera looks where the player tells it to look.
    c. It’s easier to aim ranged weapons at approaching enemies because the avatar’s body doesn’t block the player’s view and the player’s viewpoint is exactly the same as the avatar’s.
    d. Makes it easy for the player to position the avatar accurately with respect to objects. This may be required to climb stairs, pick up objects, go through doorways.
  - Disadvantages:
    a. Player can’t see the avatar, so she doesn’t have the pleasure of watching her or customizing her clothing or gear, both of which form a large part of the entertainment in many games.
    b. Player’s sense of the avatar as a distinct character with a personality and mood is reduced by not being able to see the avatar’s body language and facial expressions. Solution: scripted interactions with other characters, hints to the player or talking to herself.
    c. Designer can’t use cinematic camera angles for dramatic effect. Used in Resident Evil and Grim Fandango
    d. It makes certain types of gymnastic moves more difficult
    e. Rapid movements, especially turning or rhythmic rising and falling motions, can create motion sickness in viewers.

- Third-Person Perspective: it has the great advantage of letting the player see the avatar, and the disadvantage that it requires much more work to implement. The camera follows the avatar at a fixed distance, allowing the player to see beyond the avatar into the distance.
  - Camera behavior when the avatar turns (options):
    a. The camera always points in the direction in which the avatar looks, allowing the player to always see where the avatar is going. Since the avatar can change directions rapidly the camera must sweep around quickly in order to remain behind her, this can give the player motion sickness.
    b. The camera reorients itself behind the avatar somewhat more slowly, beginning a few seconds after the avatar makes her turn. This enables the player to see the avatar’s side and fewer players will find the images dizzying. Ex: Super Mario 64
    c. The camera reorients itself behind the avatar only after she stops moving. This is the least-intrusive way to reorient the camera. If the avatar turns back the way she came, the player will not be able see any obstacles or enemies in the avatar’s way. Ex: Toy Story 2: Buzz Lightyear to the Rescue
Player should be able to choose from Active camera mode (camera remains oriented behind the avatar at all times or reorients itself quickly) and Passive camera mode (it either orients itself slowly or only when the avatar stops moving).

- Intruding landscape objects: when a player maneuvers the avatar to stand with her back to a wall, the camera cannot retain its normal distance from the avatar.
  a. Place the camera as normal but render the wall semitransparent. This allows the player to see the world from his usual position but makes him aware of the presence of the intruding object.
  b. Place the camera immediately behind the avatar, between her and the wall, but move vertically upward and tilt it down, so the player sees the area immediately in front of the avatar from a raised point of view.
  c. Orient the camera immediately behind the avatar’s head and render he head semitransparent, until she moves so as to permit a normal camera position. Player remains aware of her position but can still see what is in front of her.

- Player adjustments to the camera: players occasionally need to adjust the position of the camera manually to get a better look at the game world without moving the avatar. Assign 2 buttons, to control manual camera movement. Buttons should make the camera circle around the avatar to the left or right, keeping her in focus in the middle of the screen. This can help with the problem of intruding landscape items.

• Aerial Perspectives: games with party-based or multipresent interaction models need a camera model that allows the player to see a large part of the game world and several different characters or units at once. Aerial perspective gives priority to the game world in general rather than to one particular character.
  - Top-down perspective: show the game world from directly overhead with the camera pointing straight down. It resembles a map, which makes it easier for players. It’s easy to implement using 2D graphics. Disadvantages: It only allows the player to see roofs and tops of people’s heads. It distances the player from the events, he feels remote from the action and less attached to its outcomes. It’s used only for showing maps or on smaller devices and web-based games. Flight Control
  - Isometric perspective: usually used to display 2D outdoor scenes. It looks across the landscape from a somewhat lower elevation, with the camera tilted down about 30 degrees from the horizontal. This perspective brings the player closer to the action than the top-down perspective and allows him to see the sides of building as well as the roofs, so the player feels more involved with the world.
  - Free-roaming camera: a 3D camera model that evolved from the isometric perspective and is made possible by modern 3D graphics engines. It allows the player considerably more control over the camera: move vertically to choose a wide or close-in view; tilt and pan in any direction at any angle, unlike the fixed camera angle of the isometric perspective. It also displays the world in true perspective. Its main disadvantage is that you have to implement all the controls for moving the camera and teach the player how to use them.
  - Context-Sensitive camera models: normally used with avatar-based or party-based interaction models. Camera moves intelligently to follow the action, displaying it from whatever angle best suits the action at any time. Disadvantages: composing a view for each location in the game world requires a lot more work and the camera can be disorienting in high-speed action situations.

• Other 2D Display Options
- Single-screen: shows entire world on one screen, normally from a top-down perspective with cheated objects. The camera never moves. *Robotron*
- Side-scrolling: consists of a long 2D strip in which the avatar moves forward and backward, with a limited ability to move up and down. The player sees the game world from the side as the camera tracks the avatar.
- Top-scrolling: the landscape scrolls beneath the avatar, sometimes at a fixed rate that the player cannot change. This forces the player to continually face new challenges as they appear at the top of the screen