The Idaho Trail

A Design Document by Adam Auer, Xingcan “Ricardo” Chen, and Sunny Qi

Prepared for CSEE4840 Embedded System Design, Taught by Steven Edwards

The Idaho Trail

Not Quite The Oregon Trail

Adam Auer, Xingcan “Ricardo” Chen, Sunny Qi
aha2197, xc2807, sq2284

17 April 2026

Contents

Introduction

In this project, we harness the capabilities of the Cyclone V FPGA to present our own take on the classic video game *The Oregon Trail*. Broadly, players will receive information on the game state through static images, simple animations, and textual information drawn to a VGA display. They will be presented with humorous scenarios ranging from lighthearted to mildly dark, whereto they will respond by entering textual commands on a USB keyboard.

The game will be a sort of "light RPG." Each day, players will be presented with a series of random events, including opportunities to rest and hunt and challenges like crossing a river. The outcomes of these situations will be influenced by a player's current health, inventory, pace, and living party members, all of which players must carefully manage if they hope to survive. Players win if they travel the total distance to Idaho and lose if they die along the way.

Outline

Input Module

- Interface with De1-SoC
- Keyboard input
    - USB interface
    - usually a single number
        - represents choice among options
    - ESC to go back
    - May allow custom character name

Gameplay Module

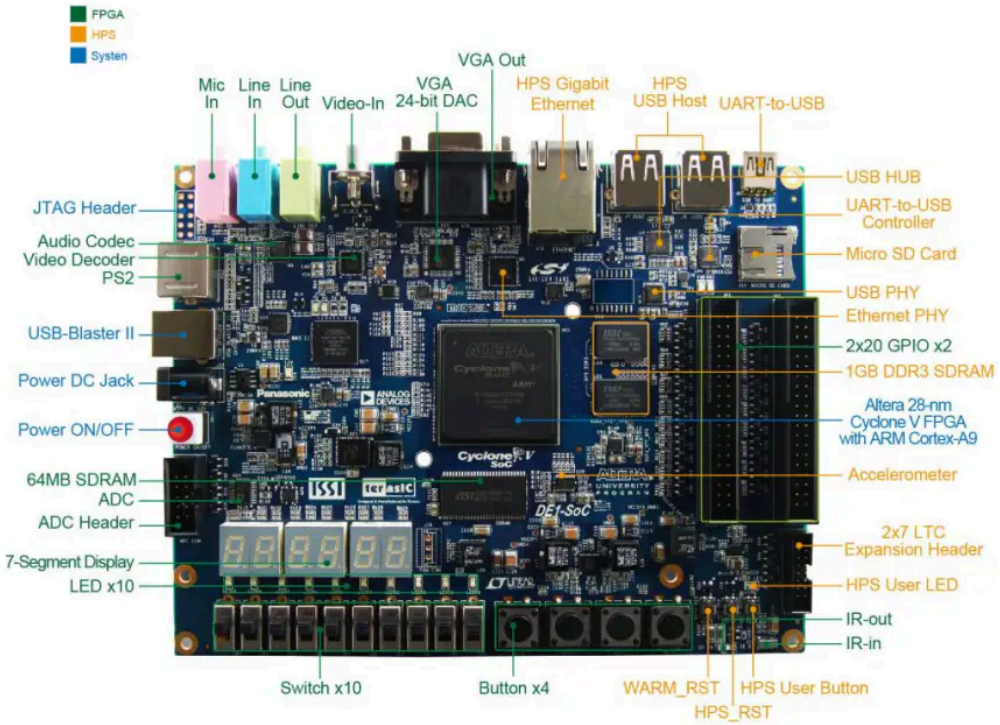
- Display split into four regions
    - Top info bar
        - Distance from next destination
        - Distance from Idaho
        - pace
        - lbs. of food
        - money
        - medicine
        - guns
        - ammunition
    - Main display
        - static image or simple animation
            - visually represents current situation
            - simple animations achieved by swapping frames back and forth
                - frames drawn ahead-of-time and loaded from files
    - Event description
        - appears when an event occurs
        - describes situation
        - presents numbered list of options

    - Input line
        - single line where a player can enter a choice in response to an event
- Gameplay mechanics
    - pace management
        - faster pace allows players to encounter fewer random, potentially game-ending events
            - that is, fewer total days on the trail
        - faster pace consumes more food
    - resource consumption
        - every day, a certain amount of food is consumed based on a player's pace and party size
    - deterministic and stochastic event resolution
        - for some events, there is a predetermined outcome based on a player's choice
            - the player, though, is not aware of the outcome before a decision is made
        - for other events, the outcome is based both on a player's decision and RNG
            - having certain party members alive or abundant supplies may increase a player's chance of avoiding a negative outcome
    - major and minor events
        - each day, a player will experience exactly **three** minor events
        - at the end of each day, a player may or may not experience a **major** event with serious consequences
            - infidelity, murder, extreme illness, etc.
    - inventory management
        - each day, if the player has covered the remaining distance to the next destination, the player may buy and sell items at the market
        - sale price always less than buy price, as is typical in popular RPG's

Display module

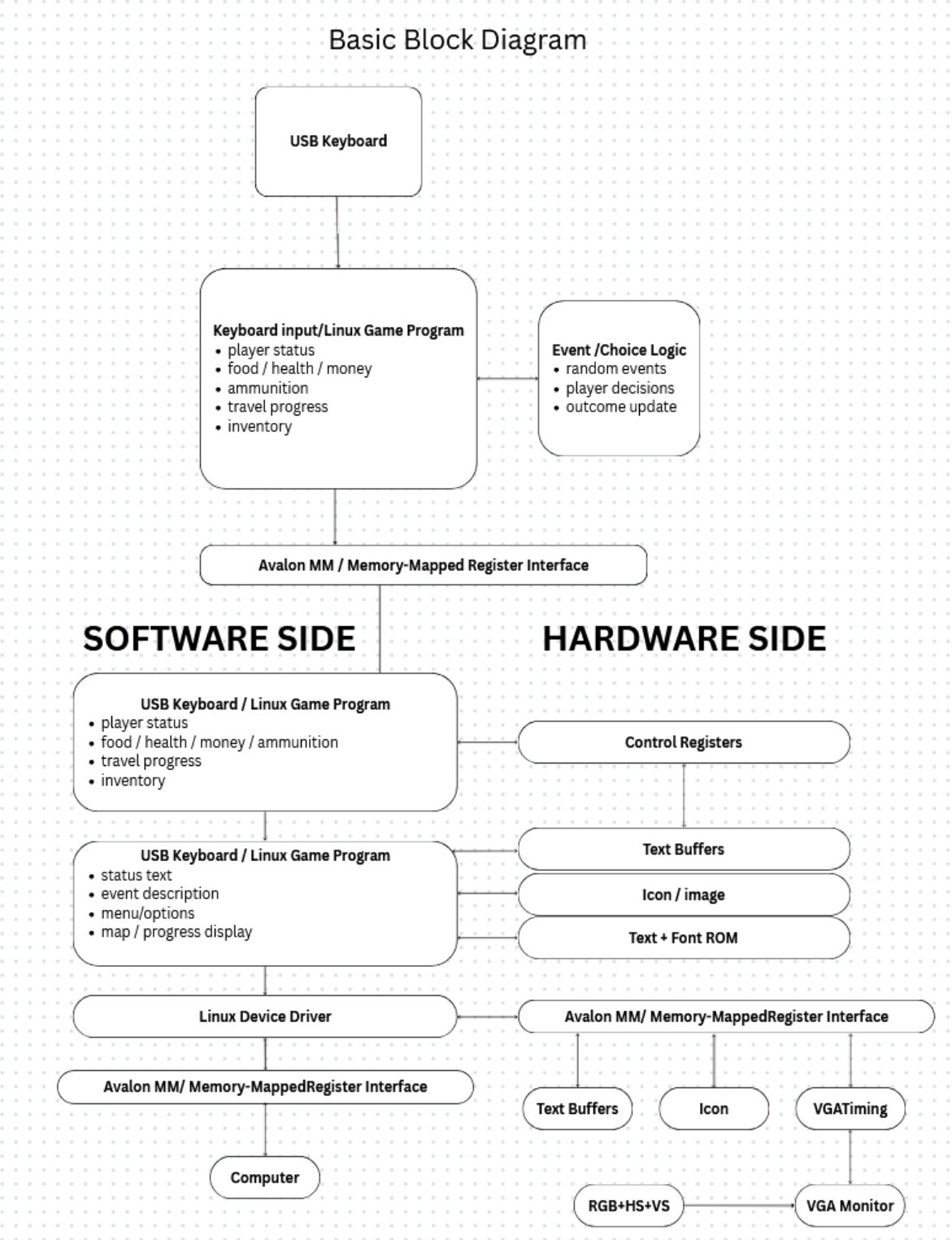
- framebuffer-based rendering
- VGA signal generation
    - ADV7123 DAC in order to generate analog VGA signal
    - display video output on LCD monitor
        - perhaps Prof. Edwards would be kind enough to provide a CRT display for a true "retro" vibe during the demonstration
- Animation
    - alternating-frame animation
        - inspired by cartoon animation
        - For example, show walking by switching between a frame with the left leg forward and a frame with the right leg forward
- Static vs. Dynamic image generation
    - central portion of the display will contain pre-drawn images
        - load pixel data from files as needed
    - top and bottom portion will display runtime-generated information including party stats and player input
        - to save memory, we shall pass ASCII characters representing this information to the hardware module and use hardware rendering to produce the final pixels

Platform

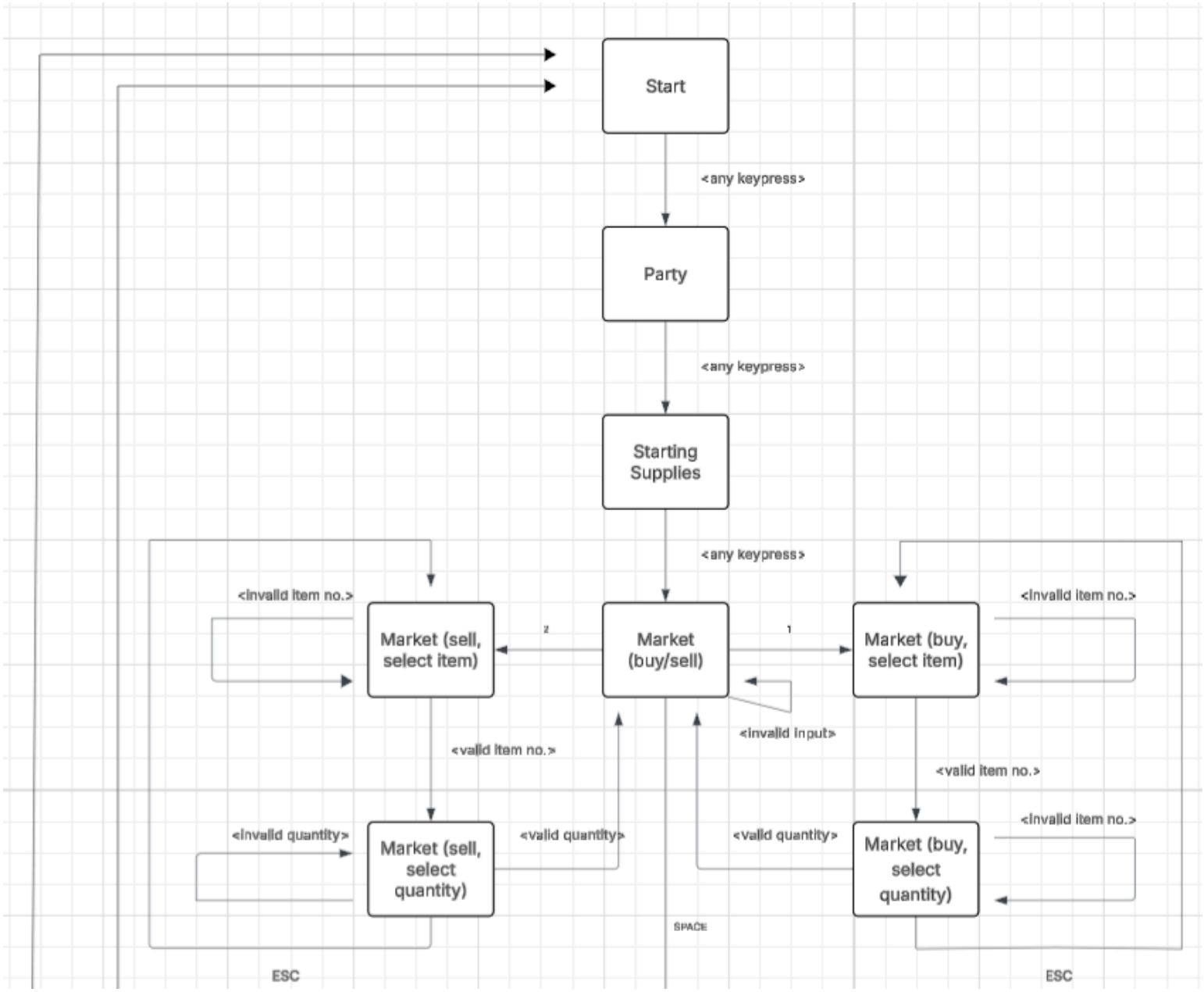


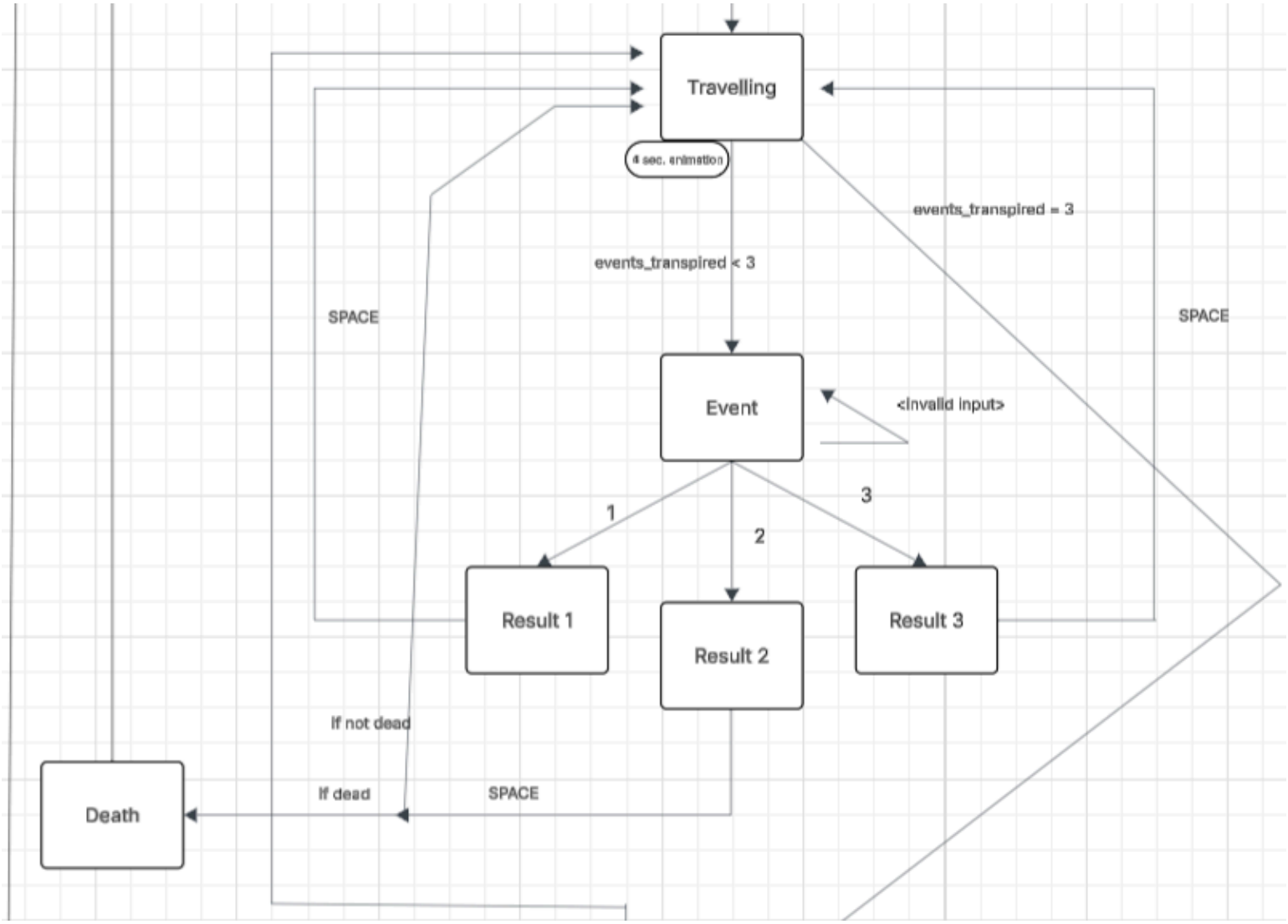
Cyclone V DE1-SoC FPGA

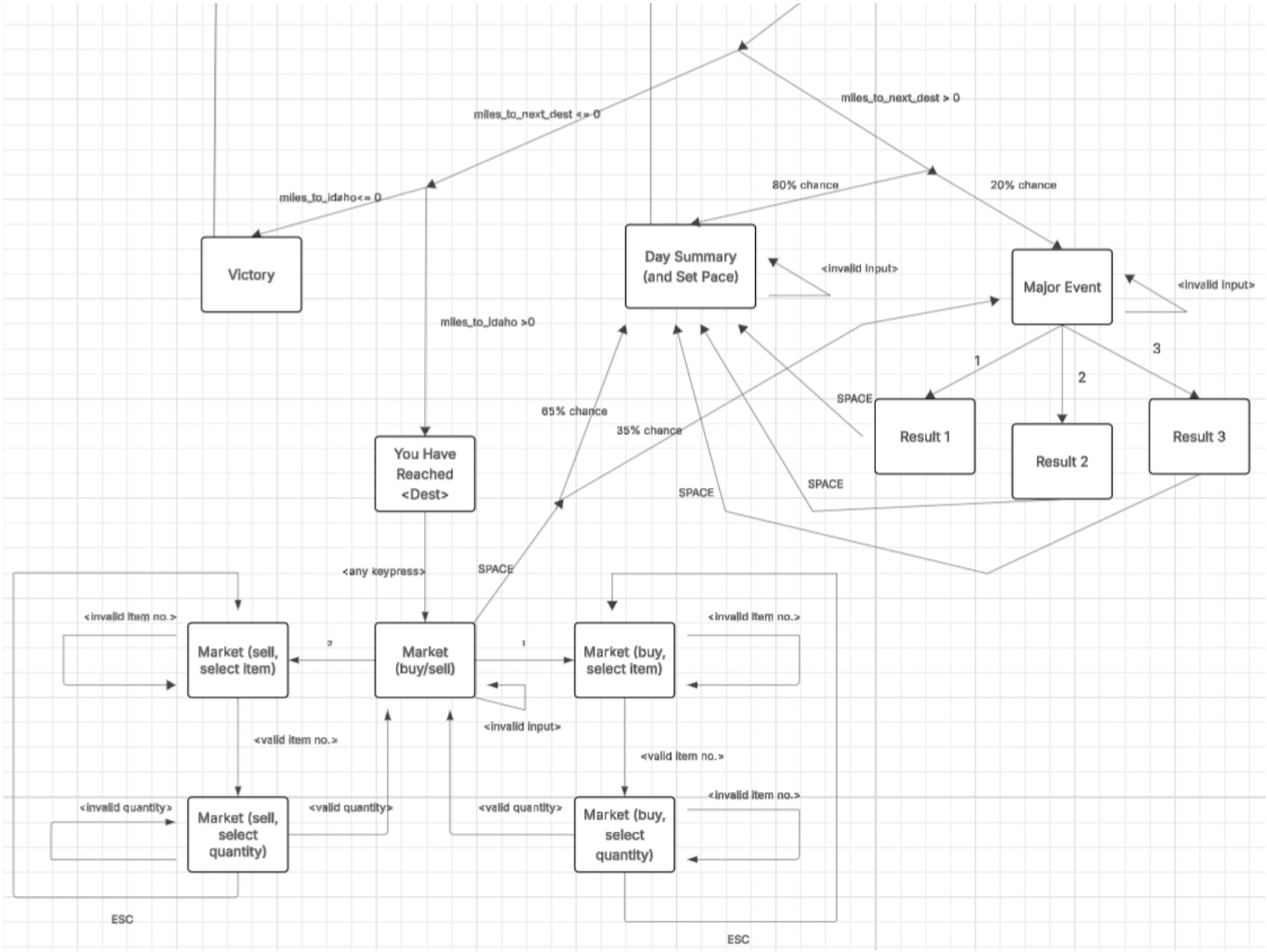
Block Diagram



Gameplay State Diagram (Full Diagram)




Travelling
4 sec. animation
events_transpired = 3
events_transpired < 3
SPACE
SPACE
Event
<invalid input>
1
2
3
Result 1
Result 2
Result 3
If not dead
If dead
SPACE
Death


miles_to_next_dest <= 0
miles_to_next_dest > 0
miles_to_idaho<= 0
80% chance
20% chance
Victory
Day Summary (and Set Pace)
<invalid input>
Major Event
<invalid input>
miles_to_idaho >0
1
2
3
SPACE
65% chance
35% chance
Result 1
Result 2
Result 3
You Have Reached <Dest>
SPACE
SPACE
<any keypress>
SPACE
<invalid item no.>
<invalid item no.>
Market (sell, select item)
2
Market (buy/sell)
1
Market (buy, select item)
<invalid input>
<valid item no.>
<valid item no.>
<invalid quantity>
<valid quantity>
<valid quantity>
<invalid item no.>
Market (sell, select quantity)
Market (buy, select quantity)
ESC
ESC

Player Input

Input Overview
We are using a USB keyboard as our main interface, primarily using the number keys to select options. For instance, to answer a yes or no question, you would type 1 for yes, 2 for no. In another example, when given 3 options, the player will use the keys from 1-3 to select one of the 3 options.

Control Signal Generation
The input controller module will monitor key presses from the USB-connected keyboard for the different numbers typed. The specific number typed is then packed and sent to the USB driver for further communication with the game engine.

Display Scheme

General Idea
To make our game fun and feel like a faithful reimplementation of *The Oregon Trail*, we need a way to draw arbitrarily complex pixel-art images to the VGA display. To give us the greatest degree of simplicity and creative freedom, we aim to develop a software/hardware interface such that a userspace C program can say "load this image file and put it on the screen." To realize this goal, we shall take inspiration from the framebuffer-based rendering approach explored in Lab 2. Our vision for our final product, however, has some peculiarities that compel us to adopt a unique rendering scheme combining aspects of both Lab 2 and Lab 3. During the main gameplay loop of "travel, event, response, repeat," the still frames of an oxcart on the plains may be generated ahead-of-time, but the party status statistics at the top of the screen, characters in the player input line, and text lines for certain events/responses must be generated at runtime. The dynamic nature of our display layout itself further complicates matters. Instead of having only one "screen" for our entire game, as in Pac-Man, we aim to have a start screen, various other introductory screens, a market screen, and a death/victory screen, all in addition to the screen for our main loop. Because the pre-generated images of each of these screens will take up a different amount of the total display area (e.g., the start screen image will fill the entire display area, and the status bar may change or be absent in the market screen), we must define different display "modes" so that the hardware can properly combine the AoT and JiT elements of each final image. We describe below these different modes and the means for communicating the necessary information to the hardware device.

Full Screen Mode: A pre-generated image fills the entire display. Used for start screen, party introduction screen, starting supplies screen, death and victory screens.

Theater Mode: A pre-generated image fills part of the screen, starting from the top row of pixels. Below this is a region of runtime-generated text describing the current situations and player options, and below this is an input line for the player to enter a response. Used for the market and major event screens.

Travel Mode: At the top of the screen is the status bar showing runtime-generated information on the player's distance to destination, pace, inventory, etc. Below this is a simple animation of pre-generated frames showing an oxcart travelling across the plains. Below this is a region of runtime-generated text describing the current situations and player options, and below this is an input line for the player to enter a response. Used for the travelling screen.

Mode Switching: The C program indicates the current mode by writing to a dedicated FPGA register, using a kernel-level device driver as an interface.

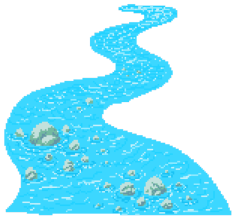
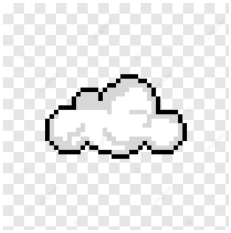
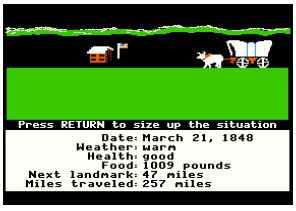
Storing and Communicating Pre-Generated Images: The pixel data for our images will be stored in a **.hex** file, wherein color data for each pixel is described by eight bits RRRGGGBB. Because we have no desire to create lavishly detailed frames and want our game to have a "retro" vibe, eight bits per pixel should suffice. At runtime, a C program will parse this file into raw binary pixel data and load the data into one of two SDRAM-backed memory buffers. To avoid tearing, the C program always writes into the region of memory that the hardware is not currently reading (i.e., we use double-buffering). All we need to pass to hardware (through the device driver), then, is the base address of each memory buffer and binary indicating the proper buffer from which to read.

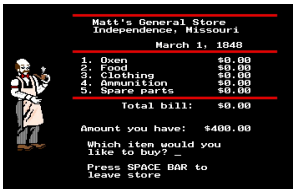
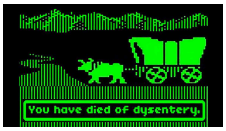
Communicating Runtime-Generated Information: To display status data and player input on the screen, the C program passes ASCII character codes (NOT raw numeric values) to dedicated FPGA registers. Character appearances are defined in a font ROM file loaded at synthesis time using a $readmemh directive.

Sprites: To make our game more visually appealing, we will, time permitting, add what we opt to call "sprites" to our travelling screen. Before loading the animation frame into SDRAM, we shall first parse the data for one or more sprites from their corresponding .hex files; then, we shall programmatically traverse the vector of animation frame data, replacing each black (#00000000) pixel in the animation frame with the corresponding pixel from the sprite logically "behind" the foreground image at that point, if one exists. For simplicity, we will design our game such that sprites themselves never overlap. (Note that the image of the oxcart is NOT a sprite; it is part of the foreground animation frame). Because this functionality is non-trivial, it will only be implemented at the end of the project work period, time-permitting.

Sprites and Frames

Size = width * length * count of sprite * 24 bits (for color)

Title	Category	Graphics	Size (bits)	# of images`	Total size (bits)
River	Sprite		40 * 100 * 24	1	96,000
Storm Cloud	Sprite		20 * 50 * 24	1	24,000
Hill	Sprite		160 * 20 * 24	1	76,800
Sun	Sprite		20 * 20 * 24	1	9,600
Title Screen	Frame		160*100*24	1	384,000
Trail (ground & oxcart)	Animated frame		160*100*24	1	384,000

Shop	Frane		160*100*24	1	384,000
Hunt (optional feature, included time-permitting)	Frame		160*100*24	1	384,000
Death	Frame		160*100*24	1	384,000

Register Map

Address Offset	Name	Width (bits)	Bit Fields
0x00	BANK	8	[0]: switch read/write pixel buffers
0x01	STATUS (read-only)	8	[0]: in vsync [1]: in hsync [2]: framebuffer ready (SDRAM has completed its last burst)
0x02	MODE	8	[2:0]: display mode, as described above (1-3)
0x03	TEXT_FG_COLOR	8	[7:5]: R [4:2]: G [1:0]: B
0x04	TEXT_BG_COLOR	8	[7:5]: R [4:2]: G [1:0]: B
0x06	(reserved)		
0x07	(reserved)		
0x08	FRAMEBUF_BASE_HI	16	[15:0]: upper framebuffer address bits (bank 1)
0x0A	FRAMEBUF_BASE_LO	16	[15:0]: lower framebuffer address bits (bank 1)
0x0C	FRAMEBUF_BANK1_HI	16	[15:0]: upper framebuffer address bits (bank 2)
0x0E	FRAMEBUF_BANK1_LO	16	[15:0]: lower framebuffer address bits (bank 2)
0x10	FIFO_THRESH	8	[7:0]: minimum fill threshold of the scanline FIFO before the VGA controller begins reading pixels; tuned to absorb SDRAM burst latency.
0x11	FIFO_STATUS	8	[7:0]: current fill level of scanline FIFO
0x14-1B	MONEY	32	[7:0]: ASCII code per cell

0x1C-1F	FOOD	32	[7:0]: ASCII code per cell
0x20-23	GUNS	32	[7:0]: ASCII code per cell
0x24-27	AMMO	32	[7:0]: ASCII code per cell
0x28-2B	LIQUOR	32	[7:0]: ASCII code per cell
0x2C-2F	MEDS	32	[7:0]: ASCII code per cell
0x30-33	TO_DEST	32	[7:0]: ASCII code per cell
0x34-37	WEATHER	32	[7:0]: ASCII code per cell
0x38-3B	TRAVELLED	32	[7:0]: ASCII code per cell
0x3C-3F	DAY	32	[7:0]: ASCII code per cell
0x40-43	HUNGER	32	[7:0]: ASCII code per cell
0x44-47	HEALTH	32	[7:0]: ASCII code per cell
0x48-4F	PACE	64	[7:0]: ASCII code per cell
0x50-9F	SIT_1	640	[7:0]: ASCII code per cell
0xA0-EF	SIT_2	640	[7:0]: ASCII code per cell
0xF0-13F	SIT_3	640	[7:0]: ASCII code per cell
0x140-18F	SIT_4	640	[7:0]: ASCII code per cell
0x190-1DF	INPUT	640	[7:0]: ASCII code per cell

Milestones

Milestone 1 (Display setup, game logic):

- Display a single frame, prompt/response interface, and information board
- Write a C program to manage the game logic
    - Interacts with the player through the terminal instead of the VGA display
- Create additional frames; allow a user program to select which frame to display

Milestone 2 (Interface setup):

- “Wire up” the game logic program to the hardware
    - Now the software can display different frames through VGA output, but player input still comes from the terminal
- “Wire up” the USB keyboard and finish text interface
    - Now the software receives input from the keyboard connected directly to the board and displays video output through the board.
    - Now the game is playable

Milestone 3 (Upgraded graphics + additional game mechanics):

- Add “sprites” so that the image of the wagon travelling across the plains is a little less boring.
- Add more frames, refine gameplay
- Make final revisions to game logic

Stretch Goal (Audio):

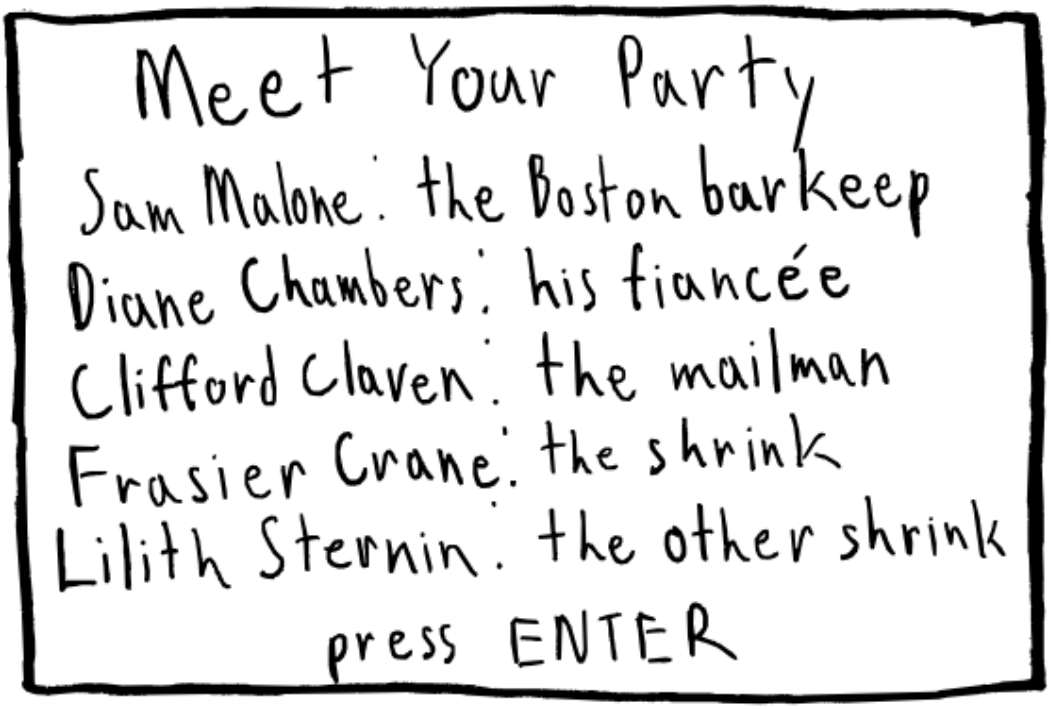
- Write appropriate Verilog to “wire up” a speaker; add code to software for beeps and boops
    - Now the user received audible as well as visual feedback
- It is very unlikely that we will get to this part, but it would make a great addition to our project

Report:

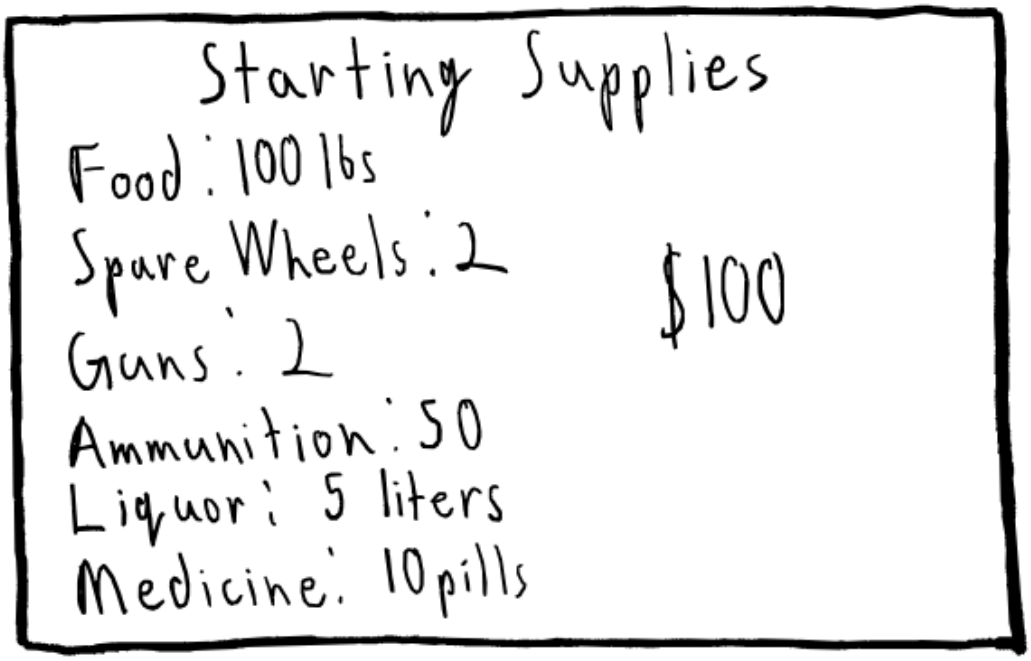
- Write a ~20 page report on the ordeal we have undergone

Concept Sketches

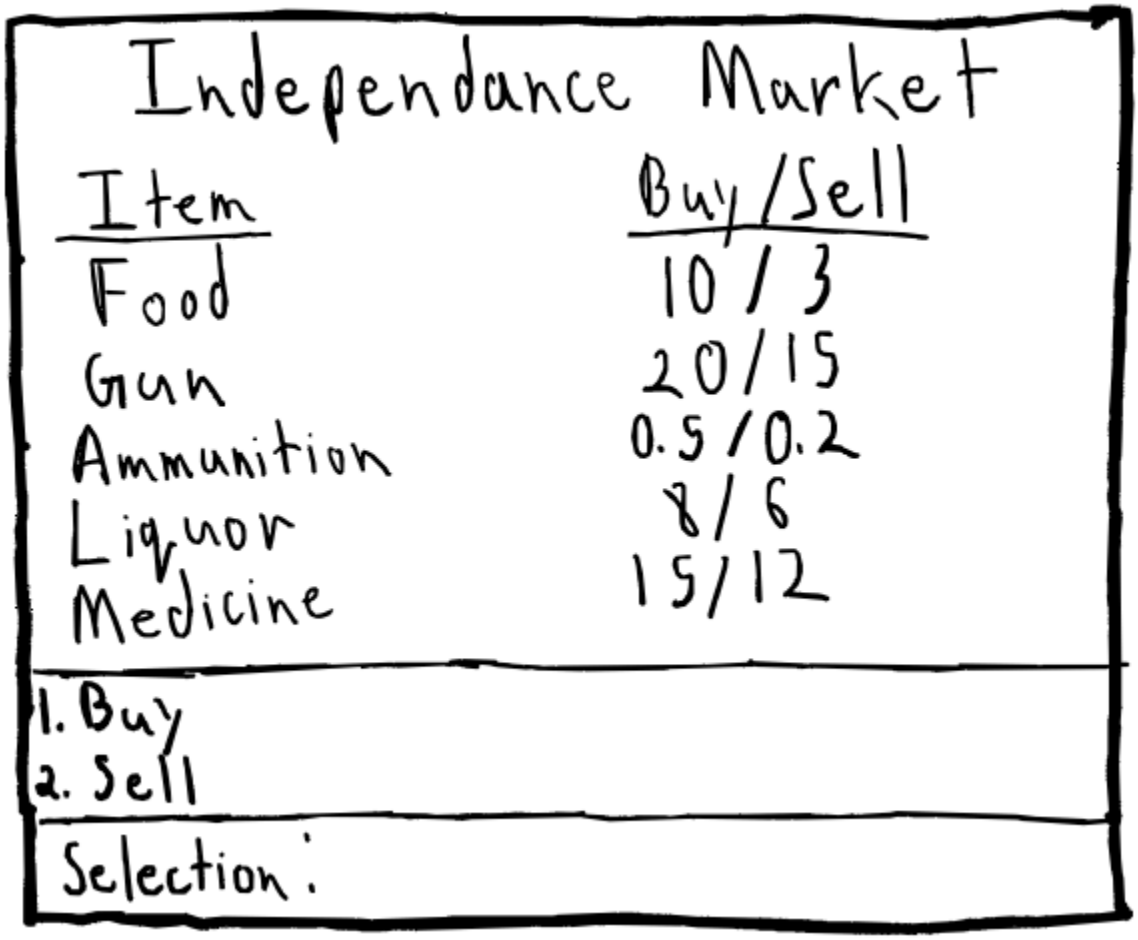
Title Screen



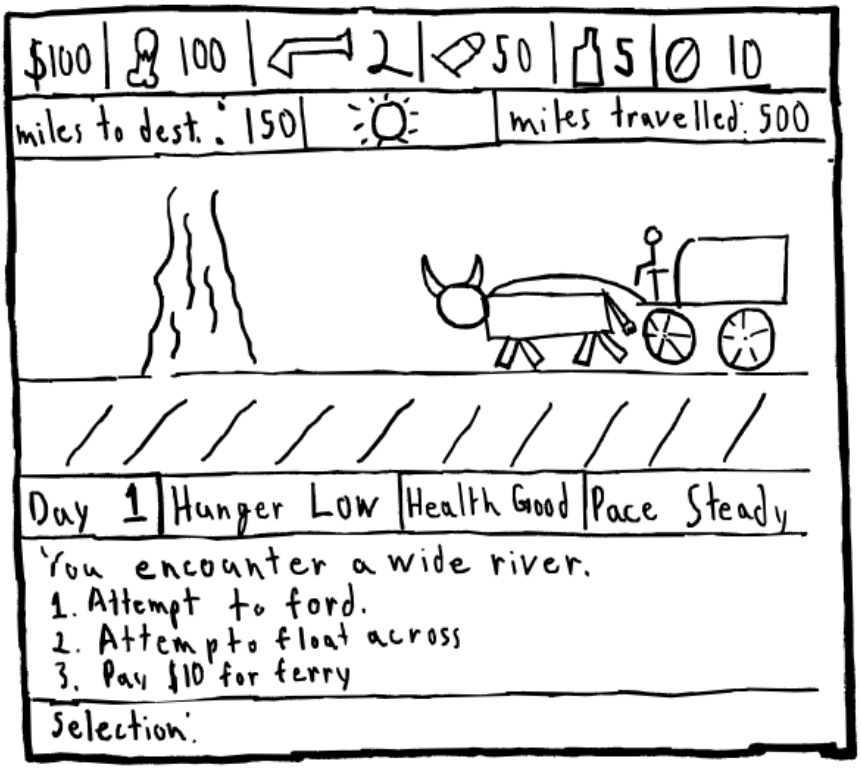
Party Introduction



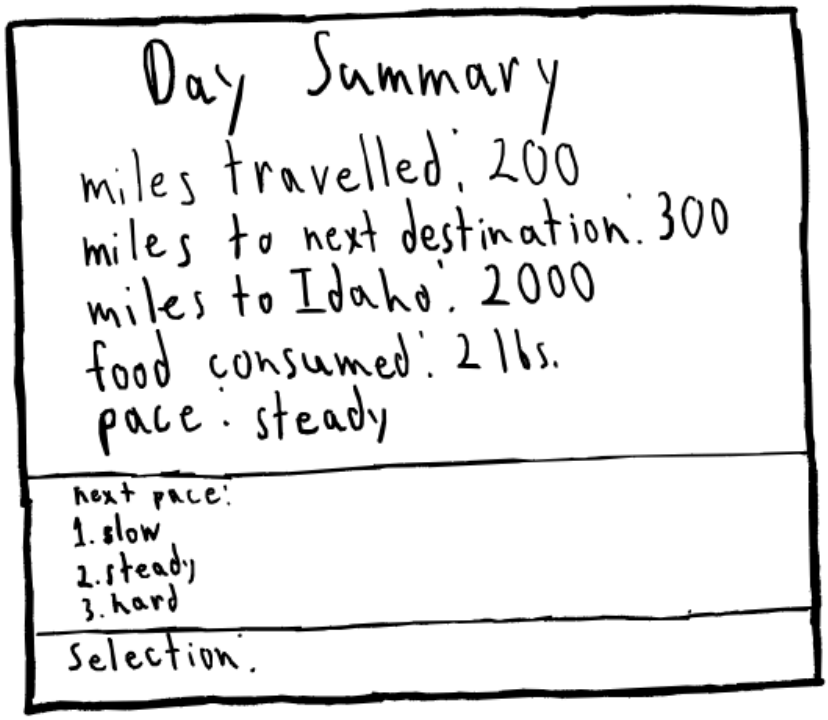
Starting Supplies



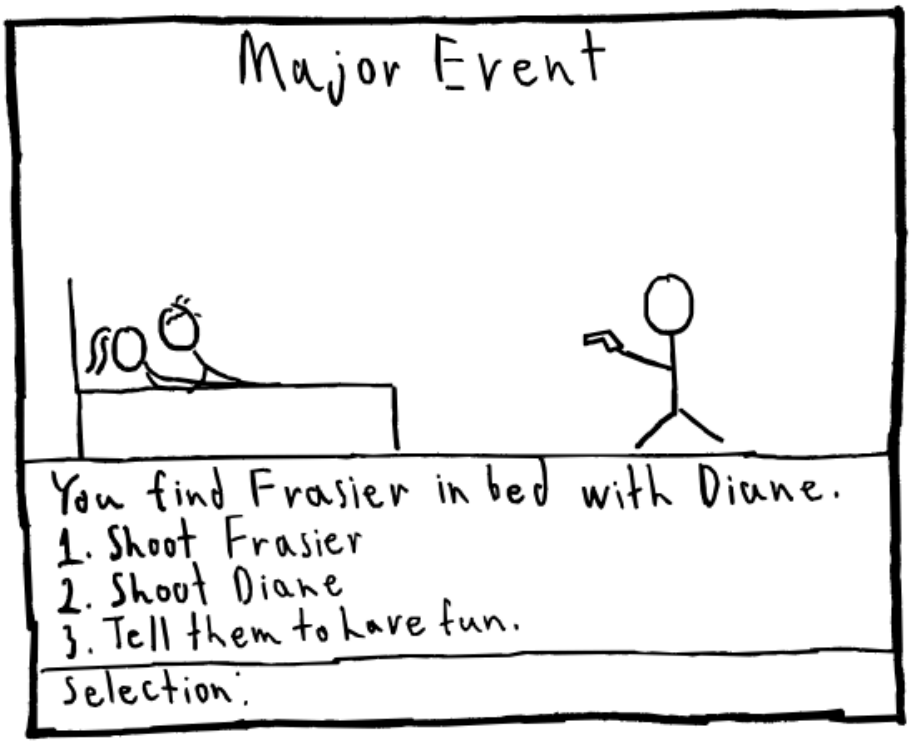
The Market



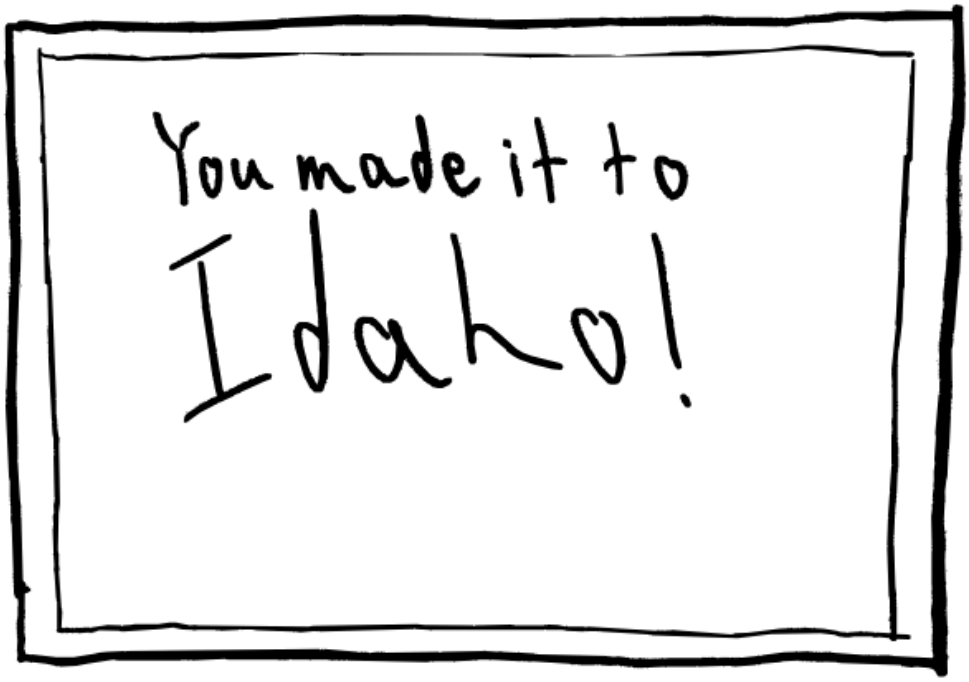
Travelling Screen



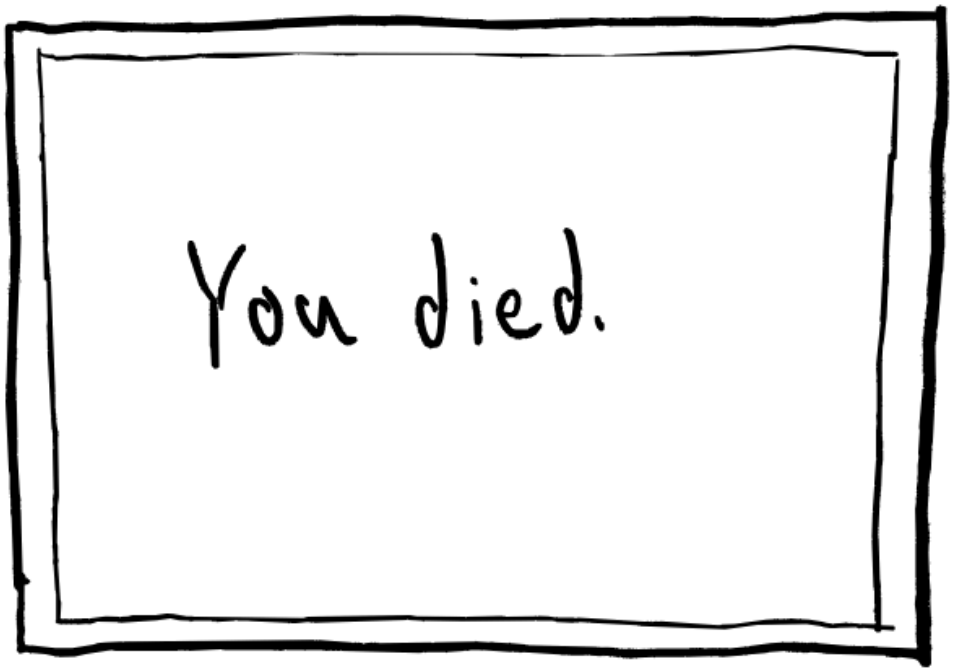
Day Summary



Major Event



Victory



Death