Developer Documentation

Overview

Software source code can be located at tippacanoe/Blackjack-Trainer. All Unity scripts were compiled in C# v.17. There are 7 script files for the whole project, these are: ModeTracker.cs, menus/MenuManager.cs, logic/Card.cs, logic/Deck.cs, logic/GameManager.cs, logics/Hand.cs, and logic/HandDisplay.cs. All scripts can be found in Assets/Scripts/.

Project Editing Requirements

- Compiler: C# v.17
- Unity Editor: 6000.2.9f1
- 8+ GB RAM
- JavaScript compatible web browser
- Unity Web Build support

Script Descriptions

Game flow is largely handled by GameManager.cs script. This manages most of how the Blackjack game flow works. It includes button management, text display, deck management, dealer plays, and hand comparisons.

ModeTracker.cs is a static data structure used to keep track of the current mode in the Blackjack scene. This is set by the main menu when the mode is selected, and determines what text (if any) is shown in the Blackjack game.

MenuManager.cs manages the menus in the main menu and enables/disables the tutorial text screen. This largely is a collection of functions that show/hide the correct buttons.

Card.cs is the card data structure used in the Blackjack scene. Every card has a prefab in Unity. The prefabs are assigned a rank, suit, and sprite through Card.cs. The suit is purely aesthetic. Rank is the value of the card (A, 2, 3, 4, etc.). Sprite is the sprite assigned to the card, and is what is shown on the screen. Deck object is a collection of the card objects.

Deck.cs is the deck data structure. This contains the linked list of card objects. It handles shuffling the deck and returning a card when cards are drawn. Additionally, deck shuffling is implemented in a quick way. The deck data structure does not *truly* shuffle ever, meaning there is never a point in the script where the code is attempting to reorganize a 52-element array at random. Instead, when a card is drawn from the deck (either by player or dealer), a random number from 0 to the length of the array is chosen, and that card is drawn from the deck causing it to be removed from the data structure. When the deck needs to be “shuffled” it is simply reset back to its default state. This is very fast.

Hand.cs is the hand data structure. This contains the list of cards in each player’s hand, and is how the game compares who wins each round. It has functionality to add cards to the list, clear the list, and counts aces correctly in all cases. It also checks if the hand count type is hard (no aces that count as 11) or soft (contains aces that count as 11).

HandDisplay.cs provides functionality to get the sprites associated with each card and show them in the relevant sections on the screen in the Blackjack scene.

Line counts for all scripts are listed below. You can find more detailed descriptions on the scripts in the developer documentation.

Script	Line Count
GameManager.cs	301

ModeTracker.cs	34
MenuManager.cs	95
Card.cs	21
Deck.cs	53
Hand.cs	142
HandDisplay.cs	73
TOTAL	719

Some scripts were longer than they currently are in their initial implementations, but I have simplified them to be shorter. This makes them easier for programmers to read while maintaining the same functionality. More lines does not necessarily mean better code, or more mechanics.

Documenting the run time of the software is somewhat difficult- this varies greatly on the system the software is running on and since the code can potentially execute indefinitely (this is a game after all, we have no control over when the player quits using the software) the run times are hard to determine. What I can say is that the software was optimized to the best of my ability and is generally very fast. Code is only executed as a direct result of player input, meaning there is no code running continuously while the player is waiting to make a decision. This minimizes resource utilization.

Unity and logic structure

There are two scenes in the game, BlackjackScene and MainMenuScene. MainMenu scene is the game scene shown when a player first starts the game. BlackjackScene is what plays

Blackjack. MainMenuScene only interfaces with the MainMenu.cs and ModeTracker.cs scripts. BlackjackScene interfaces with all scripts except MainMenu.cs.

Card prefabs are all found in Resources/Prefabs/Blackjack. Card prefabs are broken up by suit in separate folders (Spades, Clubs, Diamonds, Hearts). There is a hand prefab, which is where sprites are stored from cards in the actively played hands and allows them to be displayed on the screen.

Very little code in the scripts runs on Update in Unity, this maximizes performance as code is only executed from scripts when there is direct user input in the game. Update executes on every frame and can be very costly on performance, so this has been avoided. Instantiate is used instead of New keyword in some circumstances where prefabs are needed. This has to do with how Unity handles object initialization for prefabs, where it can instantiate but struggles to create new objects. Secure programming principles are adhered in the scripts. Variables are marked private where applicable, and can only be viewed and mutated by related getters and setters.

Functions are used in every script for logic. Every function is prefaced with a comment describing what it is for. Throughout the more complex functions, there are comments describing what certain lines of code do and why they were implemented. Camel casing is widely used across the project. Here is a table with more information on all functions used in the project:

Script	Function	Description
ModeTracker.cs	getCurrentMode()	Returns current Blackjack mode

		(standard/strategy/counting)
ModeTracker.cs	setStandardMode()	Sets Blackjack mode to standard
ModeTracker.cs	setCountingMode()	Sets Blackjack mode to counting
ModeTracker.cs	setStrategyMode()	Sets Blackjack mode to strategy
MenuManager.cs	Start()	Initializes UI
MenuManager.cs	showMainMenu()	Sets menu back to default state
MenuManager.cs	showPlayMenu()	Shows the mode selection menu
MenuManager.cs	showTutorial()	Shows tutorial text
MenuManager.cs	startStandardMode()	Starts game in standard mode
MenuManager.cs	startStrategyMode()	Starts game in strategy mode
MenuManager.cs	startCardCountingMode()	Starts game in card counting mode
Card.cs	getRank()	Returns the rank of this card object
Card.cs	getSprite()	Returns the card sprite of this card object
Deck.cs	Initialize()	Initializes deck
Deck.cs	draw()	Return and remove card from deck at random
Deck.cs	shuffle()	Return deck to base state
Deck.cs	resetDeck()	Add all cards back to deck
Deck.cs	cardsRemaining()	Check how many cards are left in the deck
GameManager.cs	Start()	Initializes the game manager and starts the Blackjack game

GameManager.cs	playerHit(bool roundStarting)	Add card to player hand and track if this is the start of the hand. If not, show strategy tooltips when relevant. Track whether player is bust
GameManager.cs	playerStand()	End player's turn and start playDealer()
GameManager.cs	dealerHit()	Add card to dealer's hand and bust their hand if over 21
GameManager.cs	playDealer()	Plays dealer in casino style. Dealer draws until they have at least 17 or are bust. Hand ends when dealer is done drawing
GameManager.cs	compareHands()	Determines the winner of the hand and displays winner in text on screen
GameManager.cs	startRound()	Starts a new round/hand for the game. This includes drawing two cards to player and dealer hands
GameManager.cs	endRound()	Updates strategy tooltips, compares hands, and shows the post round buttons
GameManager.cs	setPostRoundButtonStates(bool state)	Hides hit/stand buttons, asks if player would like to play again or return to menu
GameManager.cs	returnToMenu()	Ends Blackjack scene and returns to the main menu
GameManager.cs	updateStrategyTooltip()	In strategy mode, evaluates player's hand and dealer's known card to determine if player should hit or stand. In card counting mode, shows the currently known card count on screen to the player
GameManager.cs	increaseCount()	Increment card count

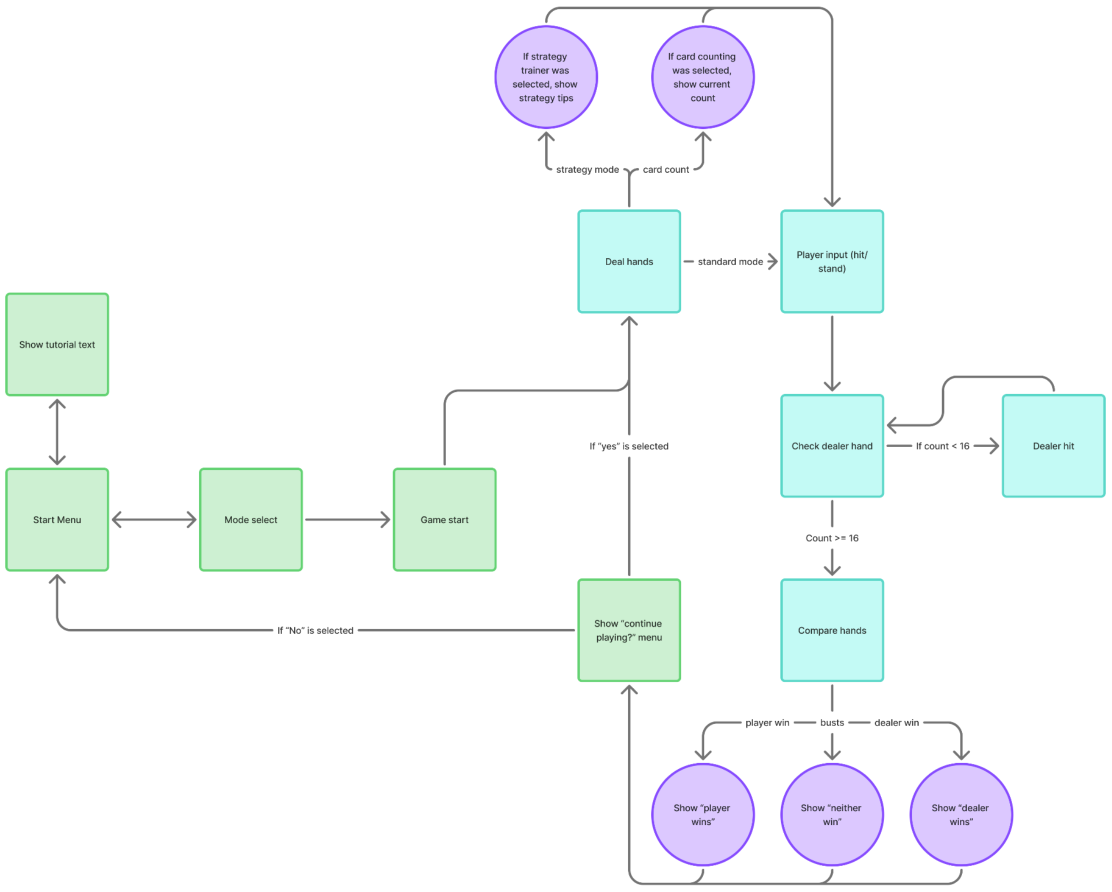
GameManager.cs	decreaseCount()	Decrement card count
Hand.cs	Initialize(bool isDealer = false)	Initializes hand and determines if this is the player's hand or the dealer's hand
Hand.cs	updateHandTotalValue()	Called after a card is drawn to hand. This determines what the hands point value is, if the hand is bust, and if the hand score type is hard/soft
Hand.cs	getHandTotalValue()	Returns the hand value
Hand.cs	getShownDealerCard()	Returns the dealer's non-hidden card
Hand.cs	getHasSoftAce()	Returns the hand score type (hard/soft)
Hand.cs	addCard(Card newCard)	Receives a card object as a parameter, and adds it to the hand. Calls updateHandTotalValue
Hand.cs	resetHand()	Removes all cards from hand and resets the state of the hand
Hand.cs	isBust()	Returns if the hand is bust or not
Hand.cs	revealHiddenCard()	If this hand is the dealer hand, it will show the dealer's hidden card to the player
Hand.cs	checkCardCount(Card card)	Checks the count of a card. 2-6 increment card count, 10 - A decrement the card count
HandDisplay.cs	Initialize(bool isDealer)	Initializes the hand display (this is how cards are shown on screen) and determines if a card must be hidden from the player
HandDisplay.cs	addCardVisual(Card card)	Display the passed card on

		screen
HandDisplay.cs	revealDealerCard(Card card)	Reveals dealer’s face-down hidden card to the player
HandDisplay.cs	positionCards()	Determines where cards will be placed on the screen
HandDisplay.cs	clearCards()	Clears cards from screen

Game flow

The dealer is an automated player. This is handled in the GameManager.cs script. When the player finishes playing (either by selecting the stand button, or busting when selecting the hit button) the player’s turn is over and the dealer plays with the playDealer function. Dealer checks to see if the hand value is >=17. If not, it will keep drawing cards until this condition is true. Once the dealer’s turn is over (hand value >=17), the dealer’s hidden card is revealed to the player, and the hands are compared. This is handled by the endRound function. When starting the round, the game checks to see if there are enough cards left in the deck (at least 10).

Also of note are how aces are counted. Aces are ranked as ‘11’ in the system. When the hand value is being calculated (updateHandTotalValue in Hand.cs), if an 11 is detected the system will only add 1 rank to the hand, and increment an ace counter. After all cards in the hand have been counted, the code will check to see if turning an ace into an 11 (instead of 1) will bust the hand value and if not, will add the remaining 10 points to the hand. It will do this for every ace.



Card counting logic follows a simple High-Low system, where 10, J, Q, K, A all count as a -1, 2 - 6 counts as a +1, and everything else does not impact the count.

We used this chart to model the strategy tooltips (credit to Blackjack Strategy Charts - How to Play Perfect Blackjack):

HARD TOTALS

	DEALER UPCARD									
	2	3	4	5	6	7	8	9	10	A
17	S	S	S	S	S	S	S	S	S	S
16	S	S	S	S	S	H	H	H	H	H
15	S	S	S	S	S	H	H	H	H	H
14	S	S	S	S	S	H	H	H	H	H
13	S	S	S	S	S	H	H	H	H	H
12	H	H	S	S	S	H	H	H	H	H
11	D	D	D	D	D	D	D	D	D	D
10	D	D	D	D	D	D	D	D	H	H
9	H	D	D	D	D	H	H	H	H	H
8	H	H	H	H	H	H	H	H	H	H

SOFT TOTALS

	DEALER UPCARD									
	2	3	4	5	6	7	8	9	10	A
A,9	S	S	S	S	S	S	S	S	S	S
A,8	S	S	S	S	Ds	S	S	S	S	S
A,7	Ds	Ds	Ds	Ds	Ds	S	S	H	H	H
A,6	H	D	D	D	D	H	H	H	H	H
A,5	H	H	D	D	D	H	H	H	H	H
A,4	H	H	D	D	D	H	H	H	H	H
A,3	H	H	H	D	D	H	H	H	H	H
A,2	H	H	H	D	D	H	H	H	H	H

PAIR SPLITTING

	DEALER UPCARD									
	2	3	4	5	6	7	8	9	10	A
A,A	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
T,T	N	N	N	N	N	N	N	N	N	N
9,9	Y	Y	Y	Y	Y	N	Y	Y	N	N
8,8	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
7,7	Y	Y	Y	Y	Y	Y	N	N	N	N
6,6	Y/N	Y	Y	Y	Y	N	N	N	N	N
5,5	N	N	N	N	N	N	N	N	N	N
4,4	N	N	N	Y/N	Y/N	N	N	N	N	N
3,3	Y/N	Y/N	Y	Y	Y	Y	N	N	N	N
2,2	Y/N	Y/N	Y	Y	Y	Y	N	N	N	N

SURRENDER

	DEALER UPCARD									
	2	3	4	5	6	7	8	9	10	A
16								SUR	SUR	SUR
15									SUR	
14										

INSURANCE OR EVEN MONEY: DON'T TAKE

KEY	H	Hit
	S	Stand
	D	Double if allowed, otherwise hit
	Ds	Double if allowed, otherwise stand
	N	Don't split the pair
	Y	Split the Pair
	Y/N	Split only if `DAS` is offered
	SUR	Surrender

How to import

Download the project directory from the Github repository. Run Unity editor, and go to the projects screen. Click Add > Add from Disk. Point Unity to the top-level project folder you downloaded. Unity will import the project and began the editor build for you.

How to build

In the editor, go to:

> File > Build Profiles > Web > Click "Open Scenes List"

> Add the MainMenu scene and ensure it is at index 0.

> Return to web build profiles. Select Build. It is going to open file explorer and ask you where to save the build. Create a new folder outside of your project for the build. Click select folder.

Unity will start building the project. This can take a while on slower systems. When it is done, you can zip the build folder, and upload it to a site like Itch.io for playback.

Licensing

The project is open source with MIT OSS license. More information can be found here: https://github.com/tippacanoe/Blackjack-Trainer/tree/main?tab=MIT-1-ov-file#readme

Misc.

All known bugs at this time are currently resolved. If more code should be implemented in the future, it should largely be built in new scripts as the existent scripts are complete. If more modes would be implemented, I would recommend pulling all of the card counting and strategy mode logic out of the GameManager.cs script and adding it to separate script files. Current build of the project is accessible on my profile on Itch.io: https://tippacanoe.itch.io/blackjack-trainer