

GDD 1200 Programming Assignment 6

XNA Blockjuck

In this assignment, you'll be playing a single hand of a game called Blockjuck. This game is something like Blackjack, but it most definitely isn't Blackjack – it's much simpler!

STARTING THE ASSIGNMENT

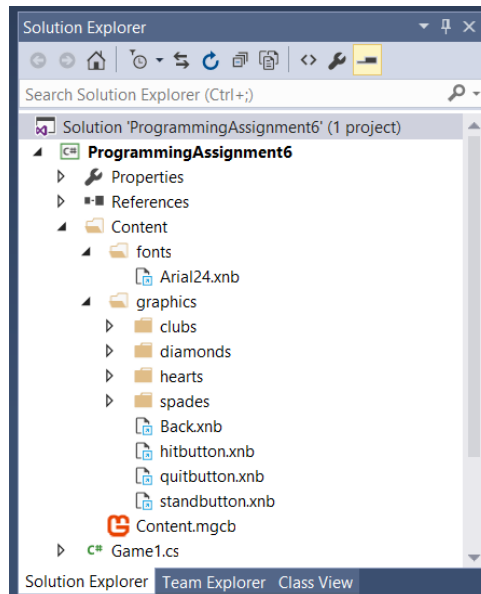
To start your work, someone in your guild should download the ProgrammingAssignment6Materials.zip file from the Programming Assignment 6 content area on Blackboard and extract the contents somewhere. The zip file contains a dll, source code, documentation, and content for the assignment.

The zip file contains documentation for the classes in the XnaCards namespace; you can use that documentation to figure out how to use those classes.

Create a MonoGame Windows Project (or MonoGame Mac Application (MonoMac) project) called ProgrammingAssignment6 in your shared Dropbox folder. DON'T call the project something else, it needs to be called ProgrammingAssignment6 for the next steps to work properly. Copy all the code from the Windows or Mac code folder from the zip file into the appropriate place (you'll have to confirm replacing the template Game1.cs file the IDE generated for you) and add all the files you just copied to the project (except Game1.cs, which is already in the project). These are all the classes you need as well as stubs for all the methods you need. I even threw in some extra code snippets where I thought they might be helpful.

Copy the dll from the zip file into the same folder as your Game1.cs file. Add the dll as a reference for the project and add a using statement for the `XnaCards` namespace to your Game1 class.

I've already used the Pipeline tool to build two separate content projects for you. Copy the XnaCardsContent and the BlockjuckContent folders from the Windows or Mac content folder from the zip file into the same folder as your Solution file. Add the content from BOTH content projects to your project. Be sure to EXACTLY mirror the folder structure that appears in the content projects; when you're done adding your content, your project should look like the image below (all the "suit" folders hold all the cards for each suit, but that doesn't fit in the image!).



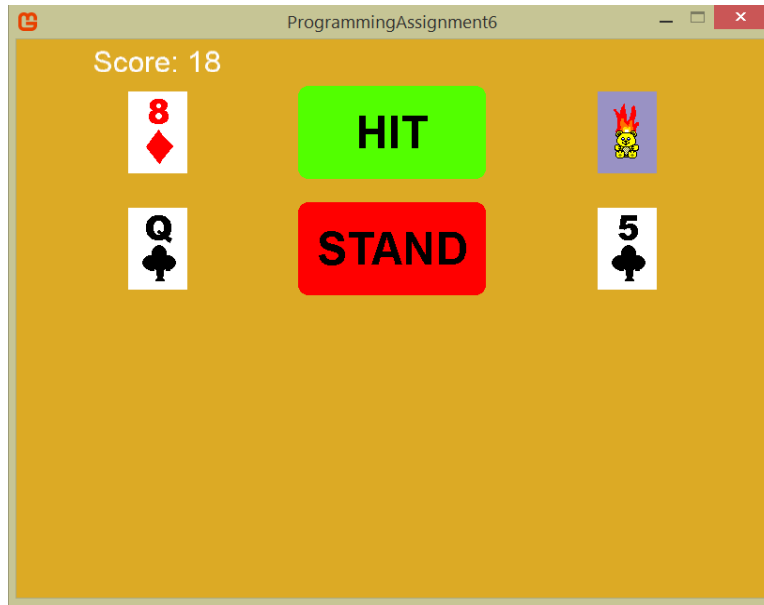
GAME BEHAVIOR

Here's how your game needs to work. The game should start with two hands: one for the player and one for the dealer. You should deal two cards from a [Deck](#) into each hand (dealing like in Las Vegas, so DON'T just give the player 2 cards then give the dealer 2 cards!). The player doesn't have to see the deal happen. Both of the player's cards should be face up (so the player can see what they have) and only the second dealer card should be face up (so the player doesn't know what the dealer has for their first card). You'll have to set the X and Y properties for the center of each card to get a reasonable display of all the cards, with the player cards on the left and the dealer cards on the right.

You'll calculate and display the score above the player's hand. You should just call the `GetBlockjuckScore` method that I've provided in the [Game1](#) class to calculate the score for the hand, but you should use the [Message](#) class I provided to actually display the score.

Finally, you'll also display two menu buttons to the right of the player cards -- one for the player to Hit (take another card) and one for the player to Stand (don't take another card). You should definitely use the [MenuButton](#) class I've provided in the project for these menu buttons.

Here's a screen shot of my solution at the start of the game (you don't have to match the spacing exactly, but it should look approximately like the screen shot):



You should carefully read the Changing Game State section below to see how the game transitions between the possible game states.

The game continues until either both the player and the dealer bust (have a hand with a score higher than `MaxHandPoints`), only the player busts, only the dealer busts, or both the player and the dealer decide to stand in a particular "turn". At that point, the dealer's hole card (the card that's face down) is flipped over, the dealer's score is displayed, the Hit and Stand buttons are removed, and a Quit button is displayed. The player clicks the Quit button to quit the game.

Here's a screen shot of my solution at the end of the game:



CHANGING GAME STATE

Finite State Machines (FSMs) are a very helpful way to specify the behavior of software (and other systems). An FSM consists of a set of states, and we move between the states along transitions. That's exactly how you'll transition between the possible game states in this game. I've provided a picture of the FSM following the description.

The game starts in the `WaitingForPlayer` state. If the player decides to hit (by clicking the Hit menu button), the game transitions to the `PlayerHitting` state. In the `PlayerHitting` state, you give the player another card, calculate and display their new score, then transition to the `WaitingForDealer` state. If the player decides to stand (by clicking the Stand menu button) in the `WaitingForPlayer` state the game transitions to the `WaitingForDealer` state.

In the `WaitingForDealer` state, if the dealer decides to hit (they have to hit on 16 or fewer points), the game transitions to the `DealerHitting` state. In the `DealerHitting` state you give the dealer another card then transition to the `CheckingHandOver` state. If the dealer decides to stand (they have to stand on 17 or more points) in the `WaitingForDealer` state the game transitions to the `CheckingHandOver` state.

In the `CheckingHandOver` state, the game checks to see if the hand is over. Hands are over when:

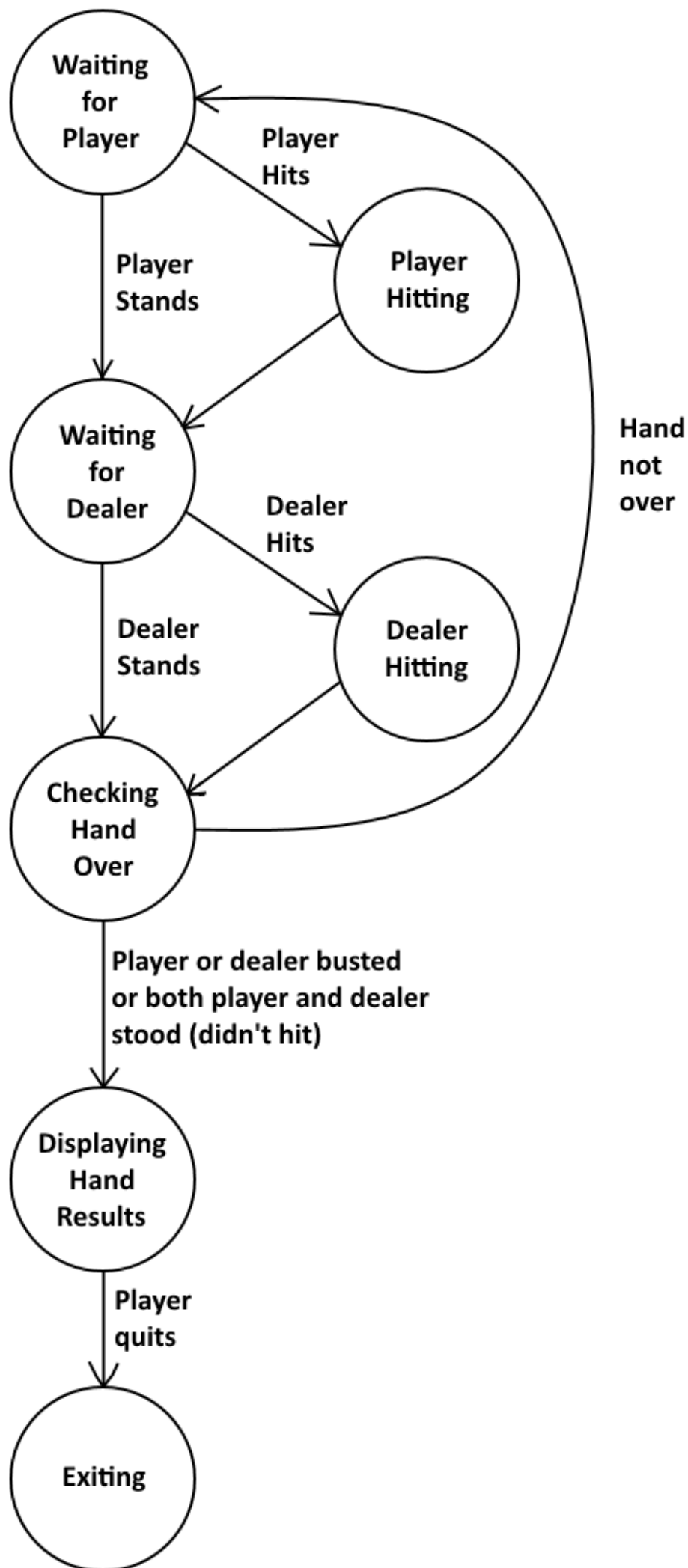
- the player or the dealer has busted (gone over `MaxHandPoints` in their hand), or
- both the player and the dealer decided to stand in a turn

If the hand is over, the game flips over the dealer's first card, creates a score message for the dealer's score, creates an appropriate winner message, hides the Hit and Stand menu buttons, creates a Quit menu button the player can use to exit the game, then transitions to the `DisplayingHandResults` state. If the hand isn't over, the game transitions to the `WaitingForPlayer` state.

Use the rule that if neither the player or the dealer has busted, the player wins if their score is highest, the dealer wins if their score is highest, and in case of a tied score nobody wins (it's a tie).

In the `DisplayingHandResults` state, if the player clicks the Quit button the game transitions to the `Exiting` state.

In the `Exiting` state, the game exits (use `Exit()` to exit the game).



REQUIREMENTS

Your solution to this problem must:

- Meet the problem specification (e.g., do what it's supposed to)
- Comply with the GDD Coding Standards

HELPFUL HINTS

Add your functionality to the game a little bit at a time. The best way to develop a game is a small piece at a time.

I provided all the fields you'll need for your solution at the top of the `Game1` class, including lots of constants you can use to properly place the objects in the game so they display properly.

Be sure to shuffle the deck before dealing the cards!

Managing the `MenuButton` and `Message` objects will be much easier if you maintain lists of them. By doing it this way, you can easily add and remove menu buttons and messages as they become active or inactive.

Your `Update` method should use a switch statement or if/else if statements to do the appropriate game processing based on the current game state.

The menu buttons in the game should only be updated in certain states, specifically in the `WaitingForPlayer` and `DisplayingHandResults` states. Make sure you do this properly.

MY IMPLEMENTATION STEPS

I implemented the required functionality in the order shown below. You do NOT have to implement your solution in this order, and I encourage you to try the assignment on your own first, but some of you might find a little extra guidance helpful as you work on your solution.

1. Set the resolution and made the mouse visible in the `Game1` constructor
2. Added code to the `Game1 LoadContent` method to create and shuffle the deck (the x and y for the deck don't really matter since the deck isn't displayed in the game)
3. Added code to the `Game1 LoadContent` method to deal both cards into the `playerHand` and the `dealerHand`. I had to pay attention to which cards should be flipped over and to set the x and y for each card so they'd be displayed properly in the game

4. Added code to the Game1 Draw method to draw the player and dealer hands. At this point, you should see the two face up player cards on the left and the one face down and one face up dealer cards on the right
5. Added code to the Game1 Draw method to tell each message in the list of messages to draw itself. At this point, you should see the player score displayed above the player cards
6. Added code to the Game1 LoadContent method to load the hit button sprite, create the hit button object and add it to the list of menu buttons. You'll need to look at the XnaCards documentation or at the MenuButton source code to see how to use the constructor properly. Because we want to move to the PlayerHitting game state when the hit button is clicked, you should pass GameState.PlayerHitting as the final argument to the constructor
7. Added code to the Game1 Draw method to tell each menu button in the list of menu buttons to draw itself. At this point, you should see the hit button, though you won't be able to do anything with it yet
8. Added code to the Game1 LoadContent method to load the stand button sprite, create the stand button object and add it to the list of menu buttons. Because we want to move to the WaitingForDealer game state when the stand button is clicked, you should pass GameState.WaitingForDealer as the final argument to the constructor At this point, you should see both buttons, though you won't be able to do anything with them yet
9. Added code to the Game1 Update method to tell each menu button in the list of menu buttons to update itself if the current state is WaitingForPlayer or DisplayingHandResults; we save the current state of the game in the currentState field. I had to get the current mouse state before telling the menu buttons to update themselves, though, because the current mouse state is a required argument for the MenuButton Update method. At this point, the menu buttons should highlight and unhighlight properly, though clicking on a button will just freeze the game with that button highlighted
10. Now it was time to start implementing the FSM, which I did with a switch statement in the Game1 Update method. Because the transitions out of the WaitingForPlayer state are handled through clicking on the menu buttons, we don't need to include that state. I added a switch statement with a case for the PlayerHitting state; the case gives the player another card, calculates the new player score, and transitions the game to the WaitingForDealer state. I had to flip the new card over and set the x and y for the card so it would be displayed properly in the game (I calculated the appropriate y location for the card based on how many cards were in the player's hand). To change the player's score, I needed to set the playerScoreMessage Text property; reference the code that I provided in the Game1 LoadContent method to create that message to see a good way to figure out the appropriate message text. At this point, you should be able to click the hit button and watch the player get a new card and a new score, then the game freezes with the hit button highlighted
11. Added a case for the WaitingForDealer state to the switch statement in the Game1 Update method. In the case, I added an if statement to implement the rules described above for deciding whether the dealer hits or stands, transitioning

the game to the appropriate state in the if and else clauses of that if statement. I got the dealer's score using the Game1 GetBlockjuckScore method. At this point, the dealer should either hit or stand based on their score, but you won't be able to see that until after the next step

12. Added a case for the DealerHitting state to the switch statement in the Game1 Update method. The case gives the dealer another card and transitions the game to the CheckingHandOver state. I had to flip the new card over and set the x and y for the card so it would be displayed properly in the game (I calculated the appropriate y location for the card based on how many cards were in the dealer's hand). At this point, you'll see the dealer either hit or stand based on their score when you get to that point in the game. I temporarily flipped over the dealer's first card while I tested this part of the code to make sure the dealer's decision was being made properly (I had to run the game a few times, of course, to check this).
13. Added a case for the CheckingHandOver state to the switch statement in the Game1 Update method. In this step, I just checked if either the player or the dealer had busted (gone over `MaxHandPoints` in their hand). Checking if the player and the dealer both stood is more complicated, so I decided to come back to that part in a later step. I added an if statement that checked if either the player or the dealer had busted, with an else part that transitioned the game to the WaitingForPlayer state if they hadn't. If one or both of them had busted (so we're in the if clause), I added an if/else if/else to check the tie/dealer busted/player busted possibilities. Within the clauses of that statement, I created the appropriate winner message text, then after that if/else if/else statement I created a new winner message and added it to the list of messages in the game. You should definitely use the Game1 messageFont and winnerMessageLocation fields that I provided to you for the second and third arguments when you call the Message constructor to create the winner message object. Outside that nested if/else if/else statement, but still in the outer if clause (because we know the hand is over), I added code to flip over the dealer's first card, created a score message for the dealer's score and added it to the list of messages in the game, removed the Hit and Stand menu buttons from the list of menu buttons in the game, created a Quit menu button the player can use to exit the game and added it to the list of menu buttons in the game, and transitioned the game to the DisplayingHandResults state. When creating the dealer's score message, you should look at the way I created the player's score message in the Game1 LoadContent method.
14. Added a case for the Exiting state to the switch statement in the Game1 Update method. Added code to the case to exit the game. At this point, you should be able to play an entire hand of Blockjuck, though the game won't end if both the player and dealer stand, so you may have to deliberately lose the game to see it end! It's possible that you'll draw so many cards that they go off the bottom of the screen before you lose; that's fine for this assignment. One way we could solve this problem would be to make sure all the cards still fit in the window after the player or dealer hits, and if not, change the Y location of each of the cards to move them closer together vertically. That's not necessary for this assignment, though

15. The final piece of functionality we need in our game is deciding that the hand is over if both the player and dealer decided to stand on a particular turn. There are a number of ways we can do this, including keeping track of the previous number of cards in both hands and comparing those to the current number of cards in each hand in the CheckingHandOver state or using bool flags that tell whether or not the player and dealer hit on their turn and checking those flags in the CheckingHandOver state. I decided to use the bool flag approach, so I included two fields in the Game1 class for these two flags. I set the playerHit flag to true in the PlayerHitting case in the Game1 Update switch statement, I set the dealerHit flag to true in the DealerHitting case in that switch statement, and I added a check for both those flags being false in the outer if statement in the CheckingHandOver case in that switch statement. I also needed to change the inner if/else if/else statement to create the appropriate winner messages in the cases where neither the player nor the dealer busted in the hand. Those Boolean expressions got a little complicated, but if you think it through carefully you should be able to figure them out. Finally, I set both flags back to false before transitioning to the WaitingForPlayer state so they'd work properly on the next turn.

TURNING IN YOUR ASSIGNMENT

This programming assignment is worth 3% of your overall course grade. **Only provide one submission per guild.**

You're required to turn in ALL of the following by the beginning of the scheduled class time on the due date:

Electronic Copy

1. Zip up your entire assignment folder into a file named <yourguildname>.zip. Log into Blackboard and submit the file into the appropriate assignment.

IMPORTANT NOTE: If your zip file doesn't contain all the required files or is zipped up using WinZip, 7Zip, or any other program different from the default Windows compression utility, you'll receive an **AUTOMATIC 0** on this assignment. Since this assignment is worth a good chunk of your overall course grade, I strongly suggest you use the Windows compression utility and check your zip file to make sure it's complete before submitting it.

LATE TURN-INS

- Turn-ins are due at the beginning of the scheduled class time on the specified due date
- No late turn-ins will be accepted