ASSIGNMENT DESCRIPTION: XNA Blackjack

In this assignment, you'll be playing a single hand of (simplified) Blackjack. You can find a description of Blackjack on wikipedia, though we'll simplify the rules a little.

To start your work, download the ProgrammingAssignment6Materials.zip file from the Required Assessment Materials course page and extract the contents somewhere.

The zip file contains a help file for the classes in the XnaCards namespace; you can use that help file to figure out how to use those classes (you can also just look at the provided source code if you prefer). Double click the XnaCards Help file to open the file. MonoGame Users: You can easily obtain free chm readers for the Mac.

If you get an error message in the right-hand pane instead of documentation links, it means you're currently blocking access to the documentation. To fix this, right-click on the XnaCards Help file on the desktop, select Properties, and select the General tab. Click the Unblock button near the lower right corner of the popup.

OK: Next, create a new XNA Windows Game called ProgrammingAssignment6; DON'T call the project something else, it needs to be called ProgrammingAssignment6 for the next steps to work properly.

OK: Replace the template Game1.cs file that was generated when you created the new project with the Game1.cs file from the code folder in the zip file.

OK: You should also copy the rest of the .cs files from the code folder in the zip file into the ProgrammingAssignment6 project folder and add them to the project by right clicking the ProgrammingAssignment6 project, selecting Add -> Existing Item ... (for MonoGame users, this is Add -> Add Files ...), navigating to the files and adding them

Adding the content to your project is a little more complicated than it has been in the past because the code I gave you for the classes in the XnaCards namespace assumes a specific folder structure in the project content.

Windows Users

OK: In Visual Studio, right click the ProgrammingAssignment6Content project and select Add -> New Folder and create a folder named Clubs. Add folders called Diamonds, Hearts, and Spades as well.

OK: In Windows, copy all the png files from the content\Clubs folder from the zip file into your ProgrammingAssignment6Content\Clubs folder.

OK: Copy the contents of the Diamonds, Hearts, and Spades folders as well.

OK: Also copy the png files (Back, hitbutton, quitbutton, standbutton) from the content folder from the zip file into your ProgrammingAssignment6Content folder.

OK: In Visual Studio, right click the Clubs folder in the ProgrammingAssignment6Content project, select Add -> Existing Item... and add all the Clubs png files to your project. Do the same for the Diamonds, Hearts, and Spades folders and the png files that aren't in one of those folders.

OK: Copy the Arial24.spritefont file from the visual studio additional content folder from the zip file into your ProgrammingAssignment6Content folder.

OK: Right click the ProgrammingAssignment6Content project, select Add -> Existing Item... and add the spritefont file to your project.

Mac Users

In MonoDevelop, right click the Assets folder, select Add -> New Folder and create a folder named Clubs. Add folders called Diamonds, Hearts, and Spades as well.

In OSX, copy all the png files from the content\Clubs folder from the zip file into your Assets\Clubs folder. Copy the contents of the Diamonds, Hearts, and Spades folders as well. Also copy the png files (Back, hitbutton, quitbutton, standbutton) from the content folder from the zip file into your Assets folder.

In MonoDevelop, right click the Assets\Clubs folder, select Add -> Add Files ... and add all the Clubs png files to your project. Do the same for the Diamonds, Hearts, and Spades folders and the png files that aren't in one of those folders.

Copy the Arial24.xnb file from the monogame additional content folder from the zip file into your Assets folder. Right click the Assets folder, select Add -> Add Files... and add the xnb file to your project.

Make sure you set the Build Properties and Quick Properties for all these assets as described in Steps 7 and 8 of the Adding Content to a MonoGame Project instructions.

Linux Users

In MonoDevelop, right click the Contents folder, select Add -> New Folder and create a folder named Clubs. Add folders called Diamonds, Hearts, and Spades as well.

In Linux, copy all the png files from the content\Clubs folder from the zip file into your Contents\Clubs folder. Copy the contents of the Diamonds, Hearts, and Spades folders as well. Also copy the png files (Back, hitbutton, quitbutton, standbutton) from the content folder from the zip file into your Contents folder.

In MonoDevelop, right click the Contents\Clubs folder, select Add -> Add Files ... and add all the Clubs png files to your project. Do the same for the Diamonds, Hearts, and Spades folders and the png files that aren't in one of those folders.

Copy the Arial24.xnb file from the monogame additional content folder from the zip file into your Contents folder. Right click the Contents folder, select Add -> Add Files... and add the xnb file to your project.

Make sure you set the Build Properties and Quick Properties for all these assets as described in Steps 7 and 8 of the Adding Content to a MonoGame Project instructions.

PEER GRADING

After submitting your work (described also in the two question parts below), you'll get the chance to grade the work of **five** of your peers. Your own work will also be assessed by your peers, from which we'll get your grade. Since you've worked hard on your submission and would like your peers to do a good job of assessing your submission, please take your time and do a good job of assessing your peer's submission in return.

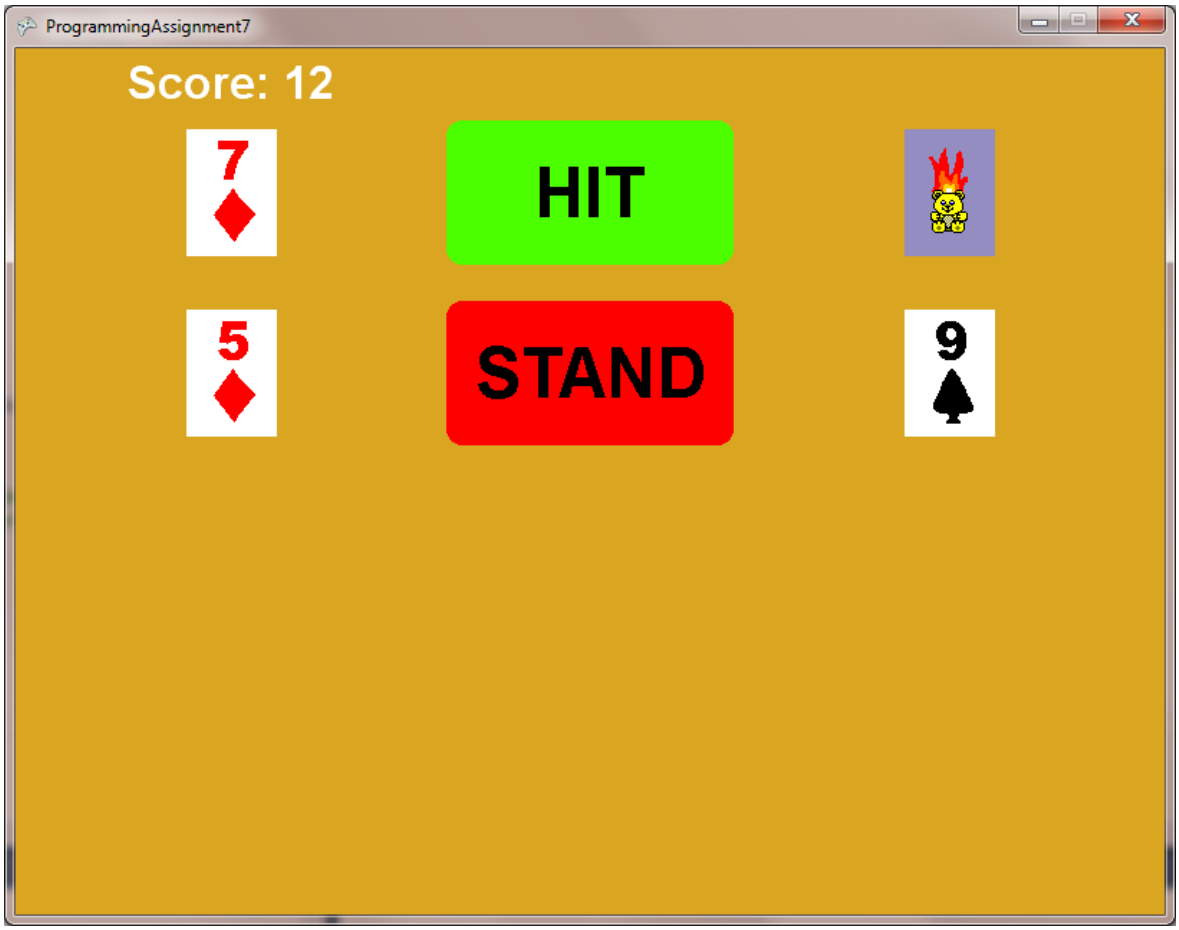
REQUIREMENTS

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 GetBlackjackScore 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 MAX_HAND_VALUE), 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:

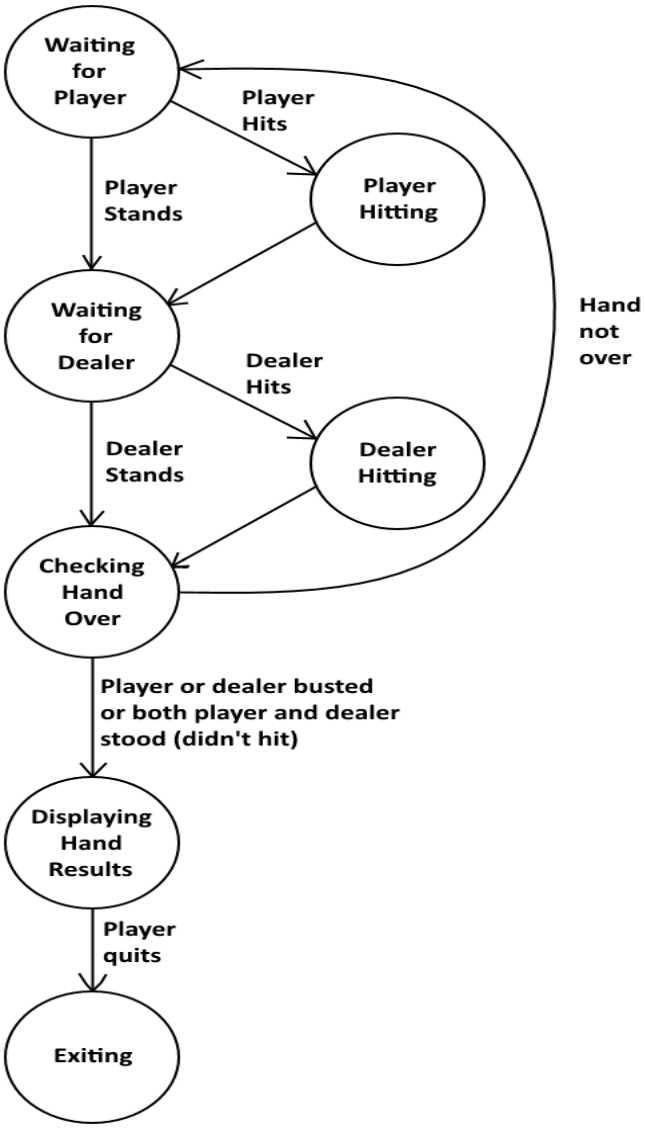
Your solution to this problem must:

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

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:

1. the player or the dealer has busted (gone over MAX_HAND_VALUE in their hand)
2. 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. In the DisplayingHandResults state, if the player clicks the Quit button the game transitions to the Exiting state.

If the hand isn't over, the game transitions to the WaitingForPlayer state.

Rule Variation: Don't worry about checking for an actual Blackjack to determine the winner. Simply 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 Exiting state, the game exits (use this.Exit(); to exit the game).

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. Note that all the code I added was in the Game1 class; I didn't make any changes to any of the other files I gave you.

OK	1. Set the resolution and made the mouse visible in the Game1 constructor
OK	2. Add 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)
OK	3. Add code to the Game1 LoadContent method to deal both cards into the playerHand and the dealerHand. 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
OK	4. Add 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
OK	5. Add 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
	Add code to the Game1 LoadContent method to
OK	6. load the hit button sprite,
OK	7. create the hit button object and
OK	8. add the hit button object 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

OK	9. Add 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
OK	10. Add 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
OK	11. Add 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
	Now it is 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. 12. I added a switch statement with a case for the PlayerHitting state;
OK	the case gives the player another card,
OK	calculates the new player score, and
OK	transitions the game to the WaitingForDealer state.
OK	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).
OK	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
OK	13. Add a case for the WaitingForDealer state to the switch statement in the Game1 Update method. In the case, add 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 GetBlackjackScore 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
OK	14. Add 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).
OK OK	15. Add 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 MAX_HAND_VALUE 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. 16. 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.
OK	17. Add a case for the Exiting state to the switch statement in the Game1 Update method. Add code to the case to exit the game. At this point, you should be able to play an entire hand of Blackjack, 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
OK	18. 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.

FOR THE STRIVERS (OR LUNATICS) ONLY

The only source code you submit for this programming assignment is the Game1 class. In Week 9, we cover how to design and implement your own classes. That means that, if you want to, you can design and implement all your own classes for the cards, deck, menu buttons, and so on. Because peer reviewers will be looking at how you interact with those classes, you should leave the method names and parameters for the methods that the Game1 class interacts with the same as I have in the provided classes, but other than that you have full freedom to build those other classes however you want to. If you've been itching to do some "heavier lifting", here's your chance!

Evaluate the video for the following 7 criteria:

1. All items displayed properly with reasonable spacing and alignment
2. Menu buttons highlight properly
3. When flip cards button is clicked, cards are flipped into battle piles and correct winner message is displayed
4. During previous step, flip cards button disappears and collect winnings button appears
5. When collect winnings button is clicked, collect winnings button disappears and flip cards button appears
6. When a player runs out of cards, only the quit button is displayed
7. When a player runs out of cards, correct winner of the game message is displayed