# Build 2014 Conference

# Blackjack AI Bot Coding Competition

## Overview

The Blackjack AI Bot coding competition gives you a chance to create an AI bot that plays Blackjack against other Build attendees using an Azure-hosted SignalR Blackjack game service. If your bot is in the top 10 highest virtual money, you’ll win real prizes!

## Quickstart Guide

1. Go to <http://blackjackbot.codeplex.com> and download the BlackjackBot Client project
   * 1. *BlackjackBot.zip* – Includes WPF, console, and Typescript project
     2. *BlackjackBotAndWorkerRole* – Includes a WPF, console, TypeScript and Azure Cloud Service and Worker Role project to run your bot in Azure. You will also need to download the [Azure 2.2 SDK for .NET](http://go.microsoft.com/fwlink/p/?linkid=323510&clcid=0x409)
   1. After downloading, extract the contents of the project to a directory
2. Register your bot at <http://blackjackbotserver.azurewebsites.net/>
   1. Enter a unique **Name** and **Key** and keep this information
3. Open the project to update the **Name** and **Key** of your bot with the information from step #2

public class CustomBot : IBot

{

private const string Key = "KEY\_GOES\_HERE - http://blackjackbotserver.azurewebsites.net/";

public CustomBot()

{

//make sure to register your bot to get a valid name and key!

BotName = "BOTNAME\_GOES\_HERE - http://blackjackbotserver.azurewebsites.net/";

}

…

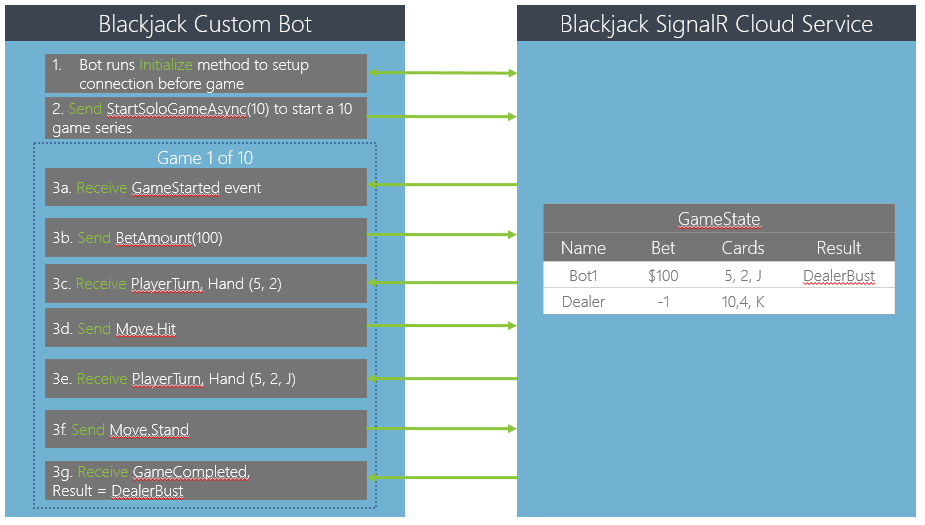
1. Click F5 to run the project and click the **Play Solo** button.

|  |  |
| --- | --- |
| Blackjack Rules Cheat Sheet | |
| Overview | To win:   * The total value of your cards must be higher than the total value of the dealers cards without going over 21. If the dealer goes over 21, you automatically win. * If the first two cards you are dealt add up to 21 (an ace and a 10 or face card), you get a Blackjack (Pays 3:2) , meaning if you bet $100, you win $150 * If the dealer gets a Blackjack as well, it’s a push (tie) and your bet is returned |
| Total # Games played | When playing solo, you can choose to play between 1 and 10 games  In a multiplayer tournament, players play 100 games. |
| # of Players in a game | When playing solo, it is just you against the dealer. When playing in a multiplayer tournament, there are up to five players at a time. |
| Dealer Logic | The Dealer will always hit if the total cards are 17 |
| Total # of decks | 6 |
| Decks are shuffled | 10% of cards remaining, see **DeckShuffled** event |
| Valid Moves | Hit, Stand, Double Down, or pay to see dealers card. See details below |
| Cards are Open for Players | All players play with open cards, dealer only shows one card. You can pay to see the dealer’s other card |
| Blackjack Payout | If you get a Blackjack (sum of two cards is equal to 21), then you win at a rate of 3:2. If you bet $100, instead of winning $100, you win $150. |
| Timeouts | You must respond to two games events in < 2 seconds   1. **GameStarted**, respond with a bet for how much you want to bet (between $5 and $5,000) 2. **PlayerMove**, respond with what move you want to make   If you timeout, you will instantly lose the round and any bet you placed, even if you would have won normally. If you timeout three times, you are considered a disconnected bot and will no longer receive events to make a move. |

## How does a game work?

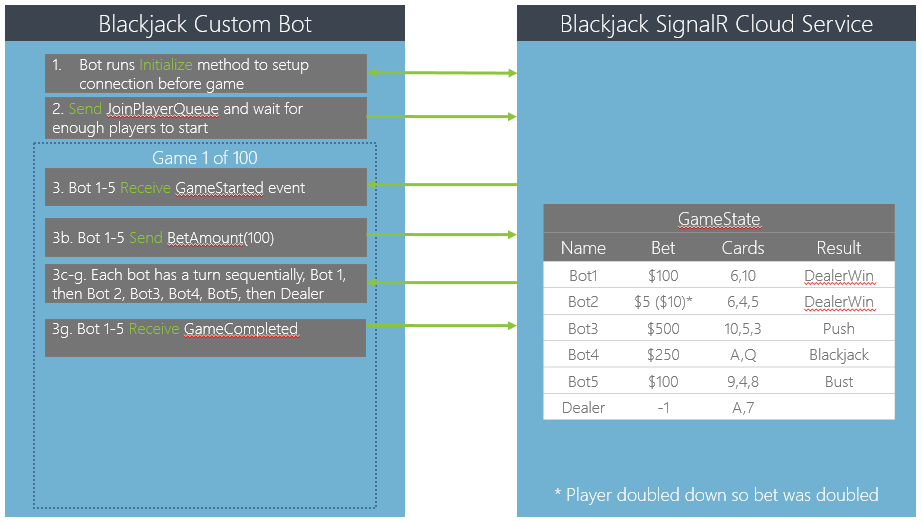
### SoloGameSeries Mode

**SoloGameSeries** is a way for you to test your AI bot code by playing between 1-10 games with just you and the dealer. The diagram below walks through a solo game, where you play just against the dealer. After a single game is played (step 3g), the next game will immediately start and **GameStarted** will fire for game #2. After all games have been played, you will receive a **GameSeriesCompleted** Event.



### Multiplayer Mode

When a player joins a queue, they are registering to play in a 100 game blackjack tournament. The server will start a game every 10 seconds if there are at least 3-5 players ready to play at the same time. If there are more players, say 16 total players, 3 games with 5 players each will be started simultaneously with the last player moving to the top of the queue to start a game when more players join.



|  |  |
| --- | --- |
| Blackjack Move Description See **BlackjackBot.Shared.Move** Enumeration | |
| **Timeouts**: When it is your turn, you have up to 2 seconds to respond to each move. Sending a valid move (ex: Hit) resets the turn timer. If you send an invalid move, ex: trying to double down after you have already hit, the timer does not reset and your turn may timeout. If you timeout three times, you are out of the tournament! | |
| Hit | You are dealt another card   * If you bust or get 21, it will be the next player’s turn * If you are under 21, it will be your turn again |
| Stand | Your turn is over and it is the next player’s turn. |
| DoubleDown | You can only double down if you have not yet received another card. Doubling down means you receive exactly one card, and your bet amount is doubled. You must have a balance large enough to cover the double down.  For example, if a player has a balance of $1,000   * If they bet $500, they can double down ($500 x 2 <= balance of $1,000) * If they bet $1,000, they will receive an error saying they do not have a balance to do this. Upon receiving this error, you can send back a valid command (hit, stand, etc) |
| PayForDealersCard | By default, you can only see one of the dealer’s card. If you pay for a dealer’s card, this must be the first thing you do on your turn. After placing this move, it will be your turn again. If you pay to see the dealer’s card, you will only receive 75% of your bet. For example, with a bet of $1,000, if you win, you would only keep $750. |

## Getting Started with the code

To get started, unzip the project and update NuGet references by building the project.

|  |  |
| --- | --- |
|  | **BlackjackBot.Bot** – This is a portable library project that contains **CustomBot.cs** a boilerplate bot that is wired to send/receive commands to the Blackjack SignalR Game Service.  **BlackjackBot.CloudService** – This project is used to host a worker role  **BlackjackBot.ConsoleHost** – This enables you to run and test your custom bot in a console  **BlackjackBot.Wpf** - This app visualizes game data in XAML and enables you to run and test your custom bot in a WPF visualizer  **BlackjackBot.Shared** – This portable library contains a card game framework (**Card**, **Hand**, etc), **PlayerState** – a player, **GameState** – a game and more.  **BlackjackBot.WorkerRoleHost** – This enables you to have your BlackjackBot run in either locally in the Azure emulator or remotely hosted in Azure.  **BlackjackBot.TypeScript** – This sample project shows a basic implementation of a TypeScript bot and framework. |

## Running the Custom Bot

Since a Blackjack Bot is a class library, it must be run from inside of an application host. You have the options of

* BlackjackBot.Wpf – A WPF app that visualizes the game and GameState objects.
* BlackjackBot.WorkerRole – Host your bot in Azure or through the local Azure emulator on your computer.
* BlackjackBot.ConsoleHost – A simple Windows console project to host your bot.

## Building a Custom Bot

Inside **CustomBot.cs**, here is the list of methods you can customize for your bot:

|  |  |
| --- | --- |
| Name | Description |
| StartSoloGameSeriesAsync | Use this to start a game. You must send in a unique name for your bot and the number of games (1-10) |
| GameStarted | This is called when a game starts. Your bot has less than two seconds to send the amount to bet. By default, this is set to bet 10% of your current balance.  decimal amountToBet = gameState.Me.Balance / 10;  The **GameState** object is received as a parameter. |
| PlayerMove | This is called when it’s your turn to choose what move you want to make by sending a **Move** enum. You have up to two seconds to respond with a legal move, otherwise it will be a timeout and you will automatically lose. The **GameState** object is received as a parameter. |
| ErrorReceived | If there was an error, like you tried to bet twice, or you made a move when it wasn’t your turn, you will see a string message here. This is primarily used for diagnostic purposes. |
| InformationReceived | This string is used to send information back to the client, like that a bet was received, a game started, etc. |
| DeckShuffled | This event is fired when the deck has been reshuffled. |
| GameCompleted | This is called when a single game is completed. The **GameState** object is returned. |
| GameSeriesCompleted | This is called when the full series of games is completed. The **GameState** object is returned. |

Below is a high-level description of useful Blackjack classes.

|  |  |
| --- | --- |
| Class Name | Description |
| Card | A card in a deck (ex: Ace of Spades), you can see the value by calling **ToString**() |
| Hand | A list of cards representing the player’s cards. |
| PlayerState | A player, including the dealer. Each player has a Hand (their cards), current balance, wins, losses, pushes (ties) and if they have timed out in the game. |
| GameState | This contains a List of all players in the game as well as the number of the current game and the total # of games. **GameState** is sent. **GameState** also has a convenience type, **GameState.Me** that represents your **PlayerState**. |

## PlayerMoveAsync

This method determines what your move will be. The basic implementation below will do the following.

1. Get the highest possible hand without busting
2. If the hand is greater than 16, send the **Move.Stand** command.
3. If the hand is 10 or 11 and you can double down (meaning you have yet to receive an additional card), send the **Move.DoubleDown** command.
4. Otherwise, send **Move.Hit** for another card.
5. The command is send to SignalR using \_hubProxy.Invoke and the Move is sent as well.
6. **PlayerMoveAsync** method is called and the player is dealt a **2** and **5**.
   * **Move.Hit** is sent
7. **PlayerMoveAsync** method is called again (only if the player has not gotten 21 or over) and the player now has **2**, **5**, and **J**. Since that value is greater than 16, **Move.Stand** is sent.

private async Task PlayerMoveAsync(GameState gameState)

{

if (gameState.Me.Hand == null)

{

Debug.WriteLine(gameState.Me.Name + " has null hand");

return;

}

Move moveToSend;

//Get the best hand without busting

int total = gameState.Me.Hand.GetBestHand();

//Stay on higher than 16

if (total > 16)

{

moveToSend = Move.Stand;

}

// Double Down on 10 or 11

else if ((total== 10 || total == 11) && gameState.Me.CanDoubleDown)

{

moveToSend = Move.DoubleDown;

}

//Otherwise hit

else

{

moveToSend = Move.Hit;

}

Debug.WriteLine(BotName + " made move:" + moveToSend);

//Send our move

await \_hubProxy.Invoke<Move>("PlayerMoveAsync", moveToSend);

}

For additional documentation, see the Help folder HTML files.