oKnow

CS 428

Final Documentation

Karan Batra

Daniel Briggs

Vikram Jayashankar

Ben Alexander

Bryan Choi

Brian Ackermann

Table of Contents

Project Overview

OKnow is a multiplayer Windows video game that combines elements of trivia with a board game atmosphere to provide tons of fun for family and friends. One to four players will attempt to be the first to traverse the game board from start to finish, answering a variety of trivia questions along the way.

To start a game, a player can customize settings about the type of board they would like to play on. There are standard configurations for three differently sized boards, but the player can choose to play on a randomly generated board, and can even customize the weights of each of the different types of tiles they would like to appear on the board. The different types of tiles represent the type of question you are asked when landing on that tile.

Once the game is started, players will choose a tile to move to by clicking on an adjacent tile to the one they are currently on. Once the tile is clicked, a question will appear and the player must answer the question correctly to move forward. The type of question is dependent on the type of tile that is clicked. There are tiles for musical questions, image questions, timed questions, and standard questions. There are also tiles with special side effects such as a death tile, which will send the player back to the start of the board if the question is answered incorrectly. If the question is answered incorrectly, the player stays at the tile they are currently on, and their number of attempts from that tile increments, causing the next question they answer to be worth less points.

These questions will be from a category selected at the beginning of the game, but can be changed if a user uses a powerup. These include swap position, skip a question, change category, and reset attempts, and can be obtained from special tiles and used to the player's advantage. The game ends once a player reaches the end of the board, at which point they will be sent back to the start menu where they can choose to start a new game, or quit the application.

Software Development Process

The process that our group followed for the project is primarily eXtreme Programming, or XP. XP is an iteration based software development methodology, which means that the tasks for the total project are split up into small portions and distributed between developers to work on for a small time period. At the end of each of these iterations, each member of the team presents his or her code and can get feedback from the users of the software as well as the other developers in the group. This quick feedback is very useful as it prevents wasting time on implementing something that is not to the user's liking. In addition to this, XP utilizes pair programming. This allows us to work together to complete iteration goals, as well as have a real time code review to catch bugs as they happen. This also makes it so if one developer is not available to answer questions about a part of the code, there is guaranteed to be another developer who worked on it and also understands how it works.

Another big aspect of XP is called test-driven development. This is useful as it forces developers to think about how the code they are going to write will be used, which in turn forces the developers to design before implementing. This also allows the developer to write some code, and then run the tests which will check if the implementation works as they expected it to. This is one key difference between our process and XP. Refactoring is also a big part of the process as it is always happening. There are always places in the code that can be improved on so that the code is either more readable, more efficient, or just more usable from a developer standpoint. Whenever we are writing new code, we always make sure to find places that we can refactor our old code so that we maintain our old code and make sure the project stays as clean and organized as possible. While we value the benefits that come with using test-driven development, we all agreed that given the time constraints that we are working with, a code-test-refactor methodology would work better for us.

The time constraints include the deadlines for the class as well as conflicts in schedules leading to limited time to meet to pair program. This means that we have less time to design, and then code, which means that it makes it more realistic to just design as we code, then write tests to make sure our implementation works as we expect it to, and then go back and refactor anything that could be implemented better.

Requirements and Specifications

User Stories

1. See the Board
    - Description: As a user, I want to be able to see the game board with its tiles and players.
    - Requirements: Have images for the tiles and players along with a layout for the board.
2. Player Movement
    - Description: As a user, I want to be able to move my piece forward a space during my turn.
    - Requirements: Taking user input and changing the player's location on the screen.
3. View a Question
    - Description: As a user, I want to be able to view a question when landing on a tile.
    - Requirements: Changing the screen from a board to a question with answers.
4. Build a Question Pool
    - Description: As a user, I want to see a different question every time I see a question.
    - Requirements: Getting a set of questions and storing them in a way we can access from the game.
5. Answer a Question
    - Description: As a user, I want to be able to answer a question when viewing it.
    - Requirements: Take user input and see if answer is correct.
6. Start Menu
    - Description: As a user, I want to be able to choose some settings such as how many players are in the game before the game begins.
    - Requirements: Display a start menu that goes to the board screen when you hit start.
7. Categories for Questions

- Description: As a user, I want to be able to set the category for questions in a game.
- Requirements: Create selector for question category and implement it.

8. Power Ups
    - Description: As a user, I want to be able to use powerups to increase my score or move distance.
    - Requirements: Creating powerups and implementing their functionality.
9. Tile Types
    - Description: As a user, I want the game board to have different tiles with different effects and different types of questions.
    - Requirements: Getting the tile images and implementing the different tiles to work correctly.
10. Player Score
    - Description: As a user, I want to be able to view my stats of correctly/incorrectly answered question as well as wins and losses.
    - Requirements: Implementing a tracking system for the different statistics.
11. Local Multiplayer
    - Description: As a user, I want to be able to play my friends and family locally on my computer. I want it to be easy for both of us to play at the same time.
    - Requirements: We are going to need add another player to the board, and keep track of each player as they progress through the board.
12. Variety of Game Boards
    - Description: As a user, I want to be able to choose or generate different boards to play on instead of the same default board every game.
    - Requirements: Implementing a either a random board generator or a system for providing multiple boards for the game to work with.
13. Sound and Music
    - Description: As a user, I want to be able to hear in game sounds and music when I'm playing.
    - Requirements: Playing sounds and music in the game.
14. Polish Screens and UI

- Description: As a user, I want to see a polished looking game with screens that have complete user interface.
- Requirements: Finalize UI designs for all screens.

15. Display Scoreboard
    - Description: As a user, I want to see the scores of all players in the game, not just one.
    - Requirements: Display a scoreboard with all players on the game screen.
16. Ranked Scoreboard
    - Description: As a user, I want to see the scores of all players in the game in descending order of score
    - Requirements: Display a scoreboard with all players on the game screen. Have a List sorter for Player total scores. Learn XNA animation or timer based object movement.
17. Better User Input
    - Description: As a user, I want to be able to interact with the game with more than just the mouse.
    - Requirements: Create listeners for other input types such as the keyboard and implement them.
18. Custom Weights for Random Tiles
    - Description: As a user, I want to be able to customize the chances of certain tiles to appear in a random board.
    - Requirements: Create UI system for modifying the chances of each tile and connect it to the random board generator.

Use Cases

- Player - Answer a Question
    - **Casual**: When a player wishes to move on the board, they will need to answer a trivia question. The player will have options for the category from which this question will come from, and will be able to select an answer from multiple choices. This answer will be given to the board which will decide the appropriate action to take based on whether the question was answered correctly or not.

- Board - Move a Player
    - **Casual**: The user will have the chance to move one space per turn. When they land on a tile they will be presented with a question if they answer that question correctly they will be allowed to move one space forward, if they answer incorrectly that will be moved one space backwards. Player movement can also be affected by power-ups. If a player uses a power up before they turn and then also correctly answer the question they have the possibility, depending on the power up, to move more than one space forward or even swap positions with another player.
- Board - End the Game
    - **Casual**: When the user correctly answers a question on the tile before the finish the player will be moved to the finish tile. On the player's next turn the player will have the opportunity to correctly answer the question presented to them on the finish tile, if they get the answer correct the player will win the game.
- Player - Start a Game
    - **Casual**: A player will start a new game by giving the system input regarding the number of players as well as the type of board that they wish to play on. The board will be generated according to these settings and then will begin the game
    - **Fully Dressed**
        - Primary Actor: Player
        - Goal in Context: To allow users to start games with the desired settings
        - Scope: System
        - Level: User Goal
        - Stakeholders and Interests:
            - Player - selects initial configuration settings
            - Board - is generated based on player settings
        - Precondition: None
        - Guarantees: A game board will be created with the number of players specified
- Player - Use a Power-up
    - **Casual**: The player selects a power-up to use from his bag. The power-up will then be applied to the board to perform some action. If it is a movement power-up, it will move a player forward. If it is a swap power-up, it will switch the

player's position with the other player. If it is a score power-up, it will double the player's score for the question.

- **Fully Dressed**
  - Primary Actor: Player
  - Goal in Context: Player wants to win the game faster by using a power-up
  - Scope: Entertainment
  - Level: User Goal
  - Stakeholders and interests:
    - Player–wants to win the game
  - Precondition: Player has to have a power-up to use
  - Guarantees: The board will only perform the action specified by the selected power-up
  - Triggers: It's the player's turn and he/she selects the power-up icon
  - Extensions: None

- Board - Retrieve a Question
  - **Casual**: The board gets the category that was selected at the beginning of the game by the players and goes to the question pool and asks for a question of that specific category. The board then uses the retrieved question to ask the player whose turn it is.
  - **Fully Dressed**
    - Primary Actor: Board
    - Goal in Context: Board is getting a question for the player to answer
    - Scope: Entertainment
    - Level: Lower-level (sub function)
    - Stakeholders and interests:
      - Player–wants a new question to answer
    - Precondition: The previous question has been answered and it's the next player's turn
    - Guarantees: The board will retrieve one question from the category selected by the player
    - Triggers: When it becomes a new player's turn

Architecture and Design

Platform and Framework

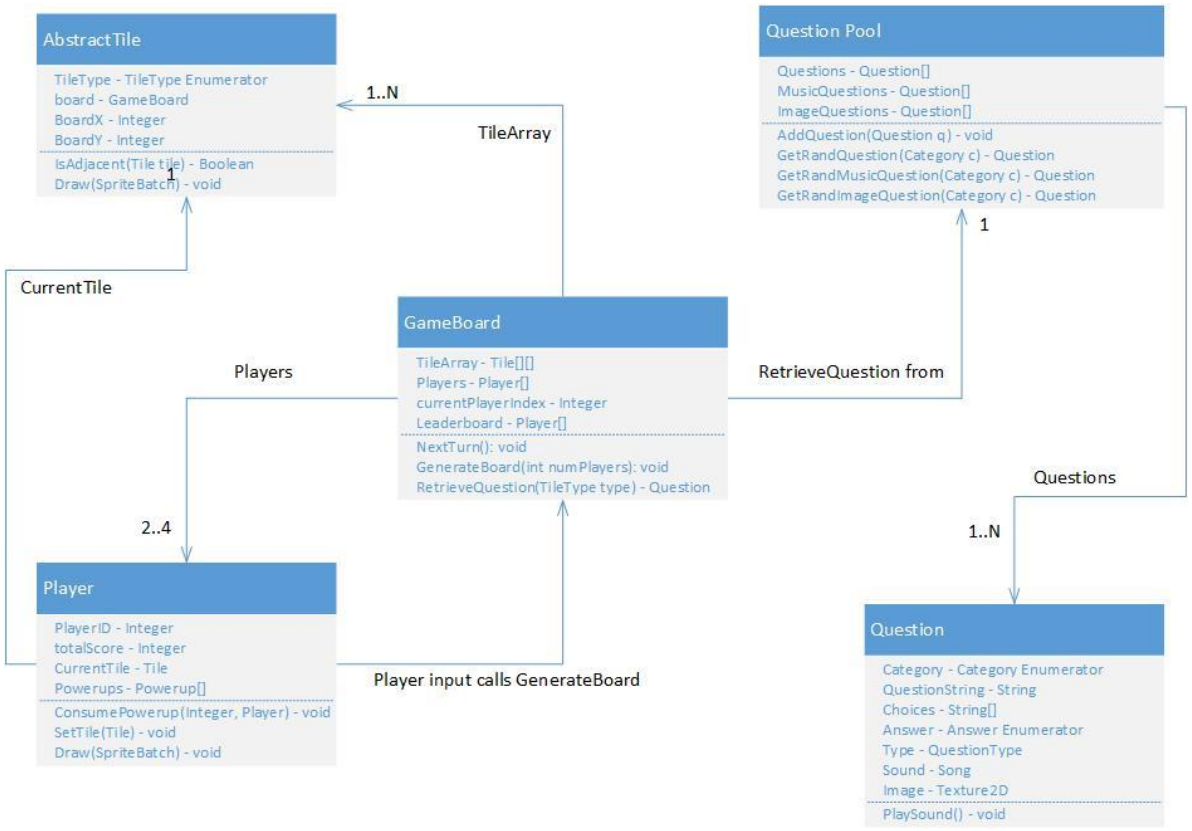
We are developing a Windows desktop game using the Ultimate edition of Visual Studio 2010. We chose this platform because we can take advantage of the XNA framework that Visual Studio provides. We were able to start our project with a game loop along with functions for easily drawing graphics to the screen. This framework let us focus our design on just the main component classes of our game, instead of worrying about content managers, playing sounds, etc… We chose the Ultimate edition of Visual Studio because it provides a diverse set testing and code analysis tools that range from basic unit testing to advanced tests such as GUI or parameter testing.

Architecture Overview

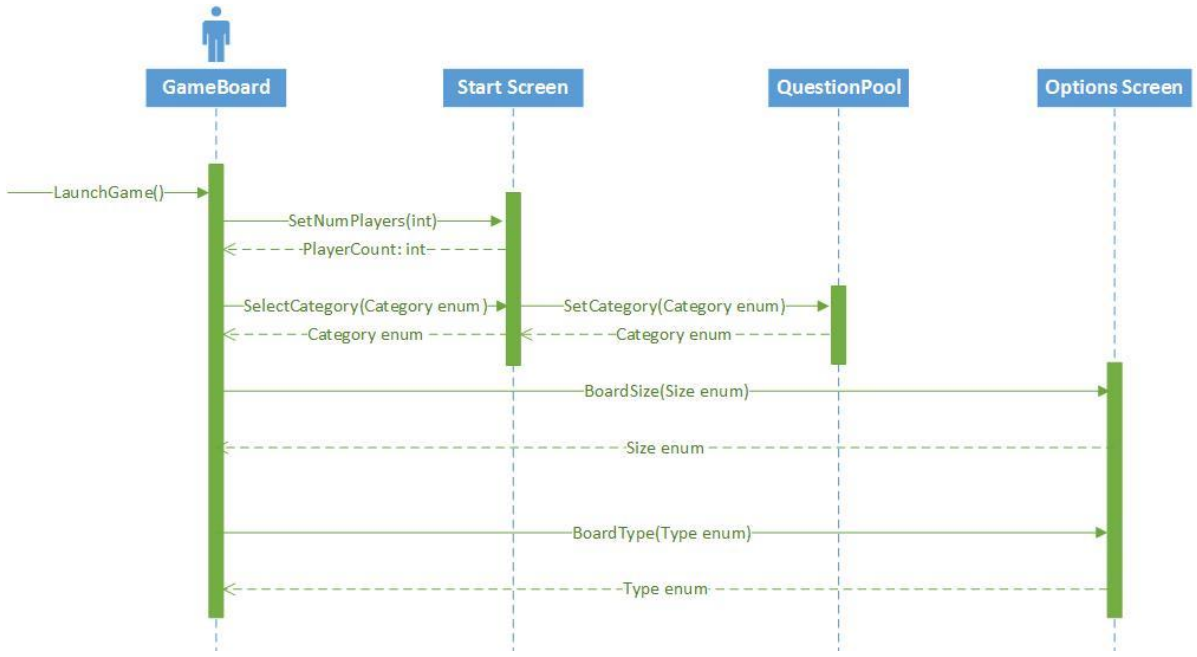
Our application is based on the model-view-controller pattern. We have a set of screens for the view; these are the `StartScreen`, `OptionsScreen`, Game screen, and Question screen. For the model, we have object classes represent each player, tile, question, and the game board. All of these classes are connected through a controlling game loop, which is in the `Game1.cs` class. This class manages the state of the game and allows the user to move freely from one screen to the next. The `GameBoard` class contains all the tiles and players of the game board. The `BoardGenerator` classes take care of generating the board, both the standard hard-coded boards and the randomly generated boards. The `Player` class contains all the information about the players of the game. The `AbstractTile` class is the parent of all the tile classes for each of the different tile types. It contains all the information tiles need. The IO classes handle different aspects of the UI for example observers, different states, and events. These classes like `rightClickEvent` and `leftClickEvent` handle when the user sends input to the game. There are also the `Question` and `QuestionPool` classes which handle how questions are generated and the classes perform the necessary checks for accuracy. Finally we have the `StaticContent` classes which help load all of the images, sounds, and fonts to be used by all of the game classes.

UML Diagrams

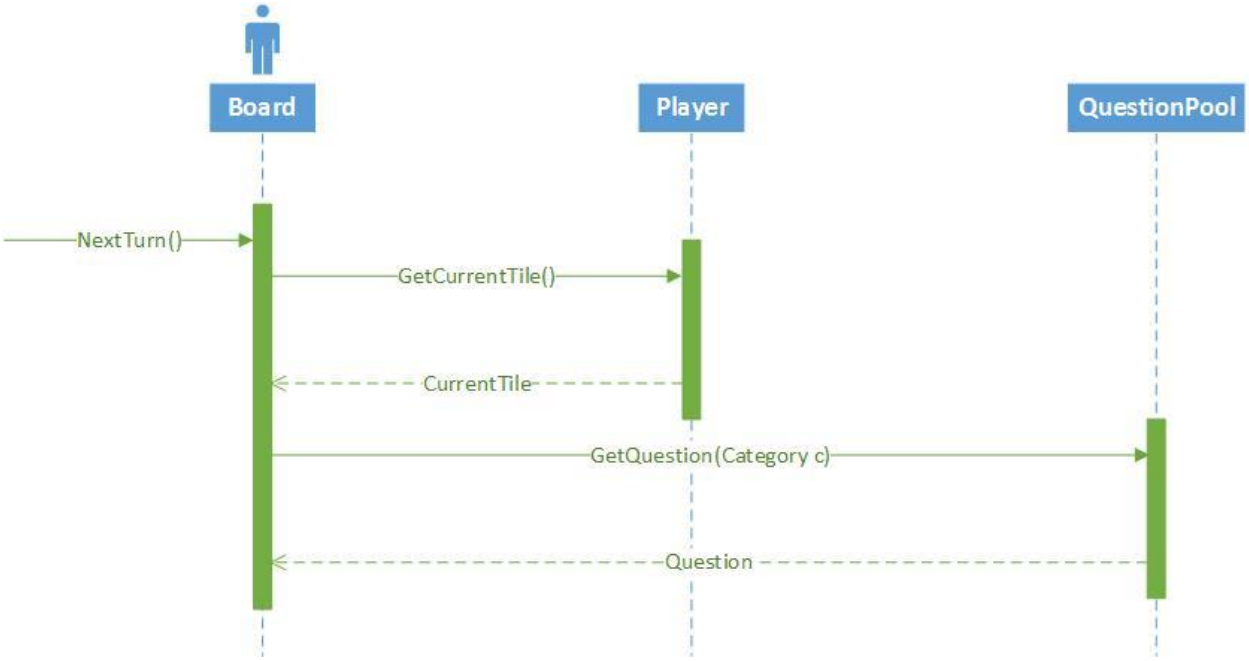
Class Diagram – GameBoard



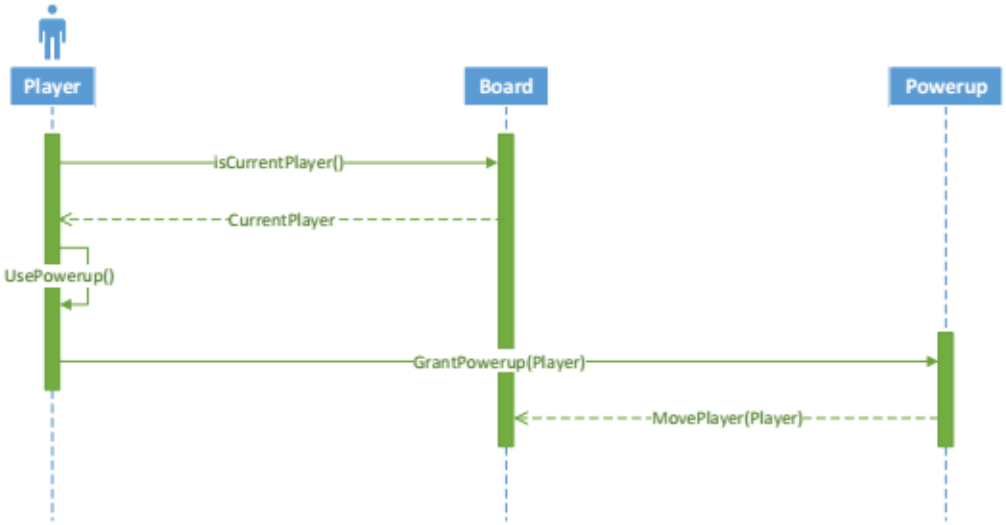
Sequence Diagram – Start a Game



Sequence Diagram – Retrieve a Question



Sequence Diagram – Use a Powerup



Future Plans

Daniel Briggs

I am very pleased with the outcome of this project. I thought we worked really well together as a team, and the result was a fun game. I enjoyed the process we used for the semester; it was very organized and made for a good experience. I would have enjoyed committing an iteration solely to refactoring because by the last week our code was quite complex. As a result, I learned the importance of thinking about the code before writing it, to ensure clean, efficient code. More forethought might have saved us quite some time on refactoring in later iterations.

Karan Batra

I had a great time working on this project. It led directly into my interests of game development, which made it a lot of fun. I really enjoyed the pair programming aspect of our software development process. Though it doesn't have a large place in most tech companies, I can definitely see the appeal of coding together. I also plan continue working on this game over the summer. The main reason is that I want to continue game development and also start working on turning it into a windows 8 or windows phone game. As for what I learned, I gained a lot of knowledge about using Visual Studio, especially its testing and code analysis tools. I also made great strides in learning more about game development, and I hope this experience can carry over when I switch to other platforms.

Ben Alexander

Oknow was the most positive experience I have ever had working on a software project with a group. A lot of our success came from our team's adherence to the process we chose. User stories enabled the team to have a common understand of what feature we needed and what the final game would look like. At the weekly meetings we would assign pairs manageable and attainable goals. Our iteration process broke down a large project into a series of small tasks which gave the project a consistent velocity. Pair programing also contributed greatly to producing working, maintainable and clear code.

Bryan Choi

This project was fun to work on and I learned a lot from my teammates. It was great to see this complex game built from the ground-up, I particularly enjoyed watching everyone pour their expertise into this project. We had all worked in smaller groups in 427, but having a bigger group like this was almost a new experience. It requires a good manager running the show, and

I'm quite proud of our project leaders for keeping us on track. While, working on this project, I learned an immense amount, not just from my partner, but everyone in the group. I had to learn an entirely new language as well as an entirely new framework for that language. But, my teammates helped ease me into it, and I had no problems contributing to this project. This was an amazing experience, and even if I never end up doing something like this again in my professional career, I can definitely see myself using C# and XNA just for fun.

Vikram Jayashankar

When I first came into this project, I had no experience with C#, any large game frameworks, or even working on developing a game at all. Most projects I've tried to start on my own have never really taken off before, so it was really great to see this one taken to completion. I really liked the process because working with a partner to write code made up for the fact that we were kind of designing as we coded. This way, we could discuss the design as we code, then write tests for the code together which is easier as well, and then refactor things that both of us felt could be done better. I think our process worked for us very well. I learned a lot for this project. I had to pick up C# as we went, which wasn't too difficult as I know Java, and the languages are pretty similar, but also got help from my teammates to learn XNA, the framework we used to build the game. On a different level, I also learned some programming techniques as some of the code I had to write during the iterations was for certain game features I've never really had to do before. This was a very enjoyable experience, and I'm glad I decided to join this team at the beginning of the semester.

Brian Ackermann

I had some experience in C# as well as Visual Studio and oKnow required someone with that knowledge and that is what brought me to this group. Besides 427 I had very little experience working with groups on large projects, especially projects that began from scratch. This project was immense and it took a group of focused individuals as well as a great team manager (Karan) took keep it from falling apart. I enjoyed working with everyone in the group and I feel we have accomplished a lot since the beginning of 428. I learned an immense amount of knowledge about C#, Visual Studio as well as the XNA framework (even if it isn't used that much anymore). I've honed my refactoring skills and sharpened other skills associated with XP style of programming and I know this will help me when working on other projects using the Agile process. Working in pairs (w/Bryan especially) taught me how to split up the work evenly so that one would not carry too much burden during each iteration. All-in-all I am glad I took 428 and met everyone working on oKnow and I am proud of what we have accomplished.

Appendix

SVN Repository - https://subversion.ews.illinois.edu/svn/sp14-cs428/_projects/oKnow/

Installation – End User

1. Run setup.exe inside oKnow/trunk/Oknow/Oknow/Oknow/publish

Installation – Developer

1. Install Visual Studio 2010 - Link
2. Download the code from the SVN repo above
3. Open the file oKnow/trunk/oKnow/oKnow.sln
4. Once Visual Studio is open, you can now browse the code
5. To run, hit the green play on the top toolbar

XML Documentation

We are using XML-style comments to document our code and generate documentation for our project. Here is an example of the XML commenting style.

```
/// <summary>
///  This class performs an important function.
/// </summary>
public class MyClass{}
```

To compile this code and generate the XML documentation, you have two options. First, you can type the following into the command line: `csc filename.cs /doc:filename.xml`

Second, in Visual Studio's Solution Explorer, right-click the project name, click Properties -> Build -> XML documentation file and enter the name of the file.