CIS*3760: Chatron - Milestone 3 Summary

Note: server may not be running at the moment, so trying to test the Slack bot or website interface, https://chatron.socs.uoguelph.ca/, may not work.

Note for Professor: For M2 corrections (discussed with Judi), please see our updated wiki documentation where we have included a more detailed explanation relating to our branch/merge strategy in reference to the M2 DevOps optional element:

- https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/wikis/TeamStandards#m2-correctional-explanation-for-devops-branchmerge-strategy

User Stories

User Story #1

https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/issues/21

No epics, Proper Format, Acceptance Criteria

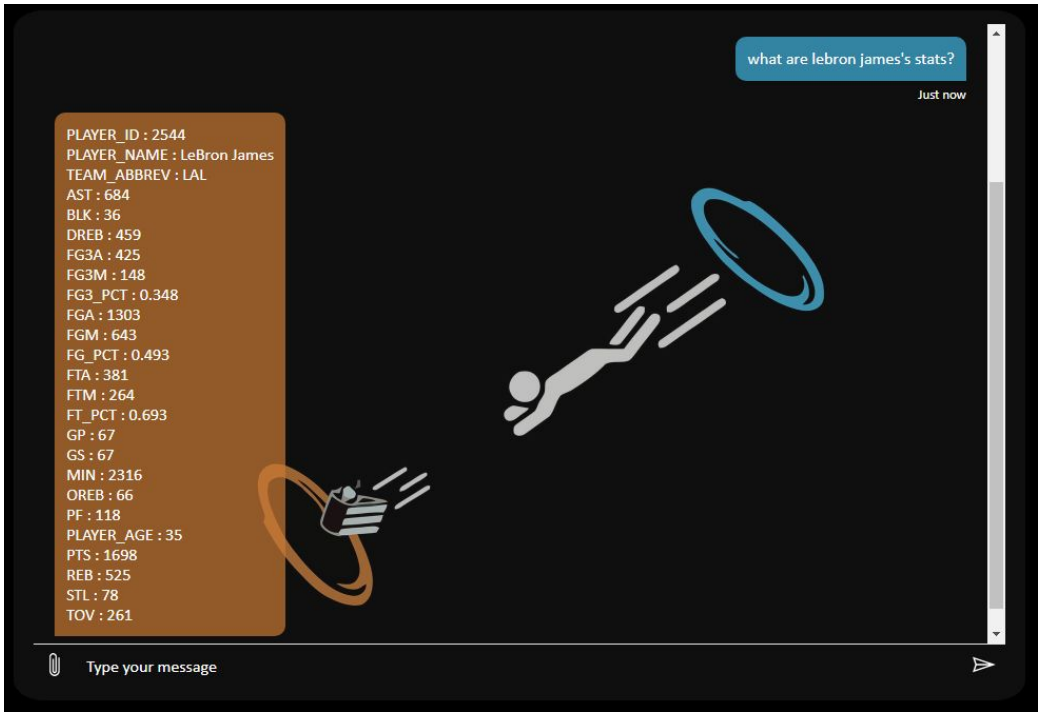
- It is a properly formatted user story: “As a <role>, I want [...] so that [...]”
- It is a broken-down task that accomplishes a single thing, i.e. not an epic
    - Part of “Epic: Language Processing” (gold label)
- It has properly formatted acceptance criteria: “Given … When … Then …”
    - As well as additional notes on desired outcome of the task
- Note there is also a weight of 8, representing an estimate of effort and time necessary
- Story also relates to #20 (https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/issues/20) because this story

is completed using the natural language processing functionality completed in that user story.

- This story also builds off of functionality developed in #6 (https://gitlab.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/issues/6)

Deliverable

- See user story acceptance criteria: https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/issues/21 and definition of done https://gitlab.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/wikis/TeamStandards#issue-board-flow
    - The client approved and accepted the delivery of basic functionality: they were able to see/use the product and receive responses to messages.
- See video in the following link under Milestone 3, User Story 4: https://gitlab.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/wikis/UserStoryDemoVideos#user-story-4

- 

Repeatable Testing

- See user story repeatable testing:

  https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/issues/21

  - All of the automated test cases implemented at the time passed when merging into master:

    https://gitlab.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/jobs/25640

```
NLP
  Send a converstional message that wouldn't be recognized
    ✓ should return a string (162ms)
    ✓ should clarify that the message was not understood (107ms)
  Analyze a conversational question for all stats for James Harden
    ✓ should return a string (186ms)
    ✓ should detect that the player was lebron james (209ms)
  Analyze a conversational question for all stats for Lebron James
    ✓ should return a string (145ms)
    ✓ should detect that the player was lebron james (172ms)
    ✓ should return the stats for lebron james (150ms)
```

  - Automated test cases were *nlp.test.js*;

        - https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/blob/master/test/nlp.test.js
- In the latest pipeline testing job, functionality for this user story was tested on lines 65-75 and 107-112 of the output: https://gitlab.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/jobs/25640

Refactored

- The code implementation for this user story has been approved by all members of the team - it is clean, refactored, and follows our coding conventions.
    - Code in *bot.js* was slightly refactored to add support for natural language processing and more robust error handling

```
} else { // the message was not a command
    const replyText = `Echo: ${ context.activity.text }`;
    await context.sendActivity(MessageFactory.text(replyText));
    const nlp = new NLP();
    try {
        const replyText = await nlp.analyze(context.activity.text);
        await context.sendActivity(MessageFactory.text(replyText));
    } catch (error) {
        await context.sendActivity(MessageFactory.text(error.message));
    }
}
```

    - All of the changes (https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/merge_requests/10/diffs) follow our coding conventions for using Pascal case when naming files and classes, using camel case when naming variable and functions, having appropriate variable names, and adding descriptive comments when necessary.

- For example, below is a descriptive comment added to the “NLP” class in *nlp.js;*

```
class NLP {
    /*
    * Function:  analyze
    * --------------------
    * analyzes a conversational query
    *
    *   message: the message sent by the user to the bot
    *
    *   returns: an array (rows from a database) with JSON object values
    *   [ {
    *       PLAYER_NAME='LeBron James',
    *       GP='119'
    *       ...
    *   } ]
    */
```

User Story #2

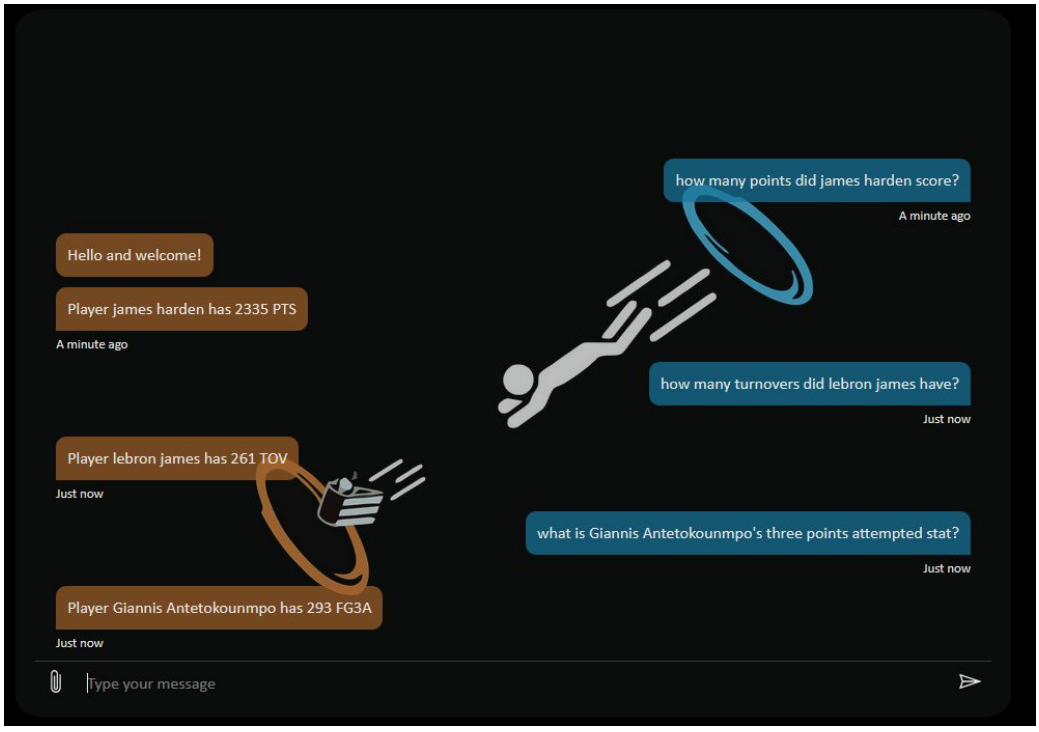
https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/issues/23

No epics, Proper Format, Acceptance Criteria

- It is a properly formatted user story: “As a <role>, I want [...] so that [...]”
- It is a broken-down task that accomplishes a single thing, i.e. not an epic
    - Part of “Epic: Language Processing” (gold label)
- It has properly formatted acceptance criteria: “Given … When … Then …”
    - As well as additional notes on desired outcome of the task
- Note there is also a weight of 8, representing an estimate of effort and time necessary
- Story also relates to #20 (https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/issues/20) because this story is completed using the natural language processing functionality completed in that user story.

- This story also builds off of functionality developed in #12 (https://gitlab.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/issues/12)

Deliverable

- See user story acceptance criteria: https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/issues/23 and definition of done https://gitlab.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/wikis/TeamStandards#issue-board-flow
    - The client approved and accepted the delivery of basic functionality: they were able to see/use the product and receive responses to messages.
- See video in the following link under Milestone 3, User Story 5: https://gitlab.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/wikis/UserStoryDemoVideos#user-story-5
- 

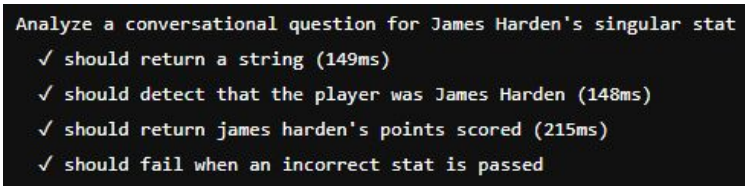
Repeatable Testing

- See user story repeatable testing:

  https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/issues/23

  - All of the automated test cases implemented at the time passed when merging into master:

    https://gitlab.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/jobs/25687

  - Automated test cases were *nlp.test.js*;

    - https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/blob/master/test/nlp.test.js

- In the latest pipeline testing job, functionality for this user story was tested on lines 77-81 and 113-119 of the output:

  https://gitlab.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/jobs/25687

Refactored

- The code implementation for this user story has been approved by all members of the team - it is clean, refactored, and follows our coding conventions.

  - All of the changes (https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/merge_requests/11/diffs) follow our coding conventions for using Pascal case when naming files and classes, using camel case when naming variable and functions, having appropriate variable names, and adding descriptive comments when necessary.

  - For example, below is a new variable named appropriately and descriptively with camel case;

    ```
    const stat = wit.retrieveStat(response);
    ```

User Story #3

https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/issues/38

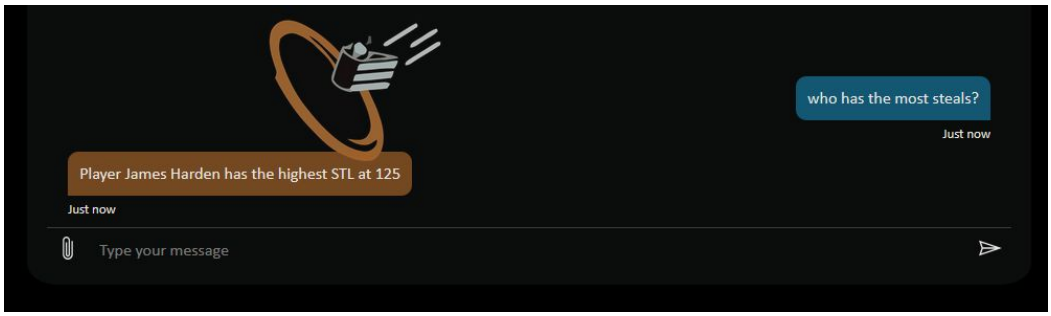
No epics, Proper Format, Acceptance Criteria

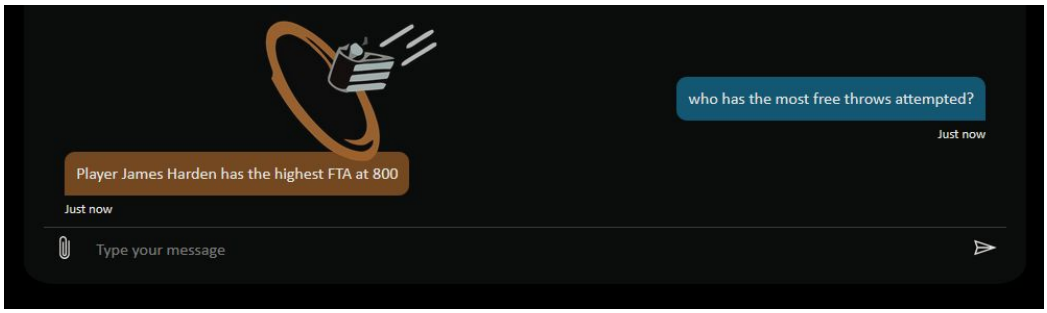
- It is a properly formatted user story: “As a <role>, I want [...] so that [...]”
- It is a broken-down task that accomplishes a single thing, i.e. not an epic
    - Part of “Epic: Language Processing” (gold label)
- It has properly formatted acceptance criteria: “Given … When … Then …”
    - As well as additional notes on desired outcome of the task
- Note there is also a weight of 8, representing an estimate of effort and time necessary
- Story also relates to #20 (https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/issues/20) because this story is completed using the natural language processing functionality completed in that user story.
- This story also builds off of functionality developed in #13 (https://gitlab.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/issues/13)

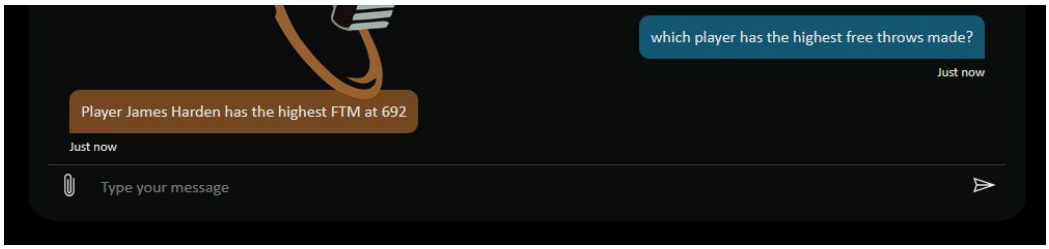
Deliverable

- See user story acceptance criteria: https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/issues/38 and definition of done https://gitlab.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/wikis/TeamStandards#issue-board-flow
    - The client approved and accepted the delivery of basic functionality: they were able to see/use the product and receive responses to messages.

- See video in the following link under Milestone 3, User Story 6: https://gitlab.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/wikis/UserStoryDemoVideos#user-story-6

- 

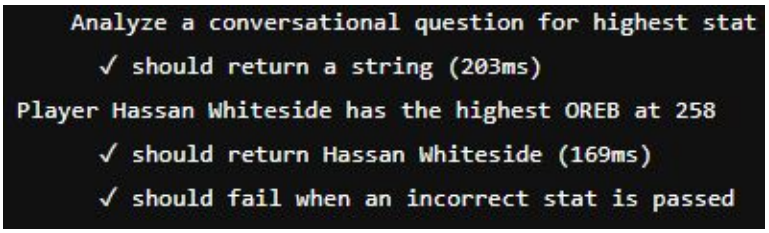
- 

- 

Repeatable Testing

- See user story repeatable testing: https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/issues/38
    - All of the automated test cases implemented at the time passed when merging into master:

https://gitlab.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/jobs/25721



- Automated test cases were *nlp.test.js*;
  - https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/blob/master/test/nlp.test.js

- In the latest pipeline testing job, functionality for this user story was tested on lines 81-85 and 117-121 of the output: https://gitlab.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/jobs/25721

Refactored

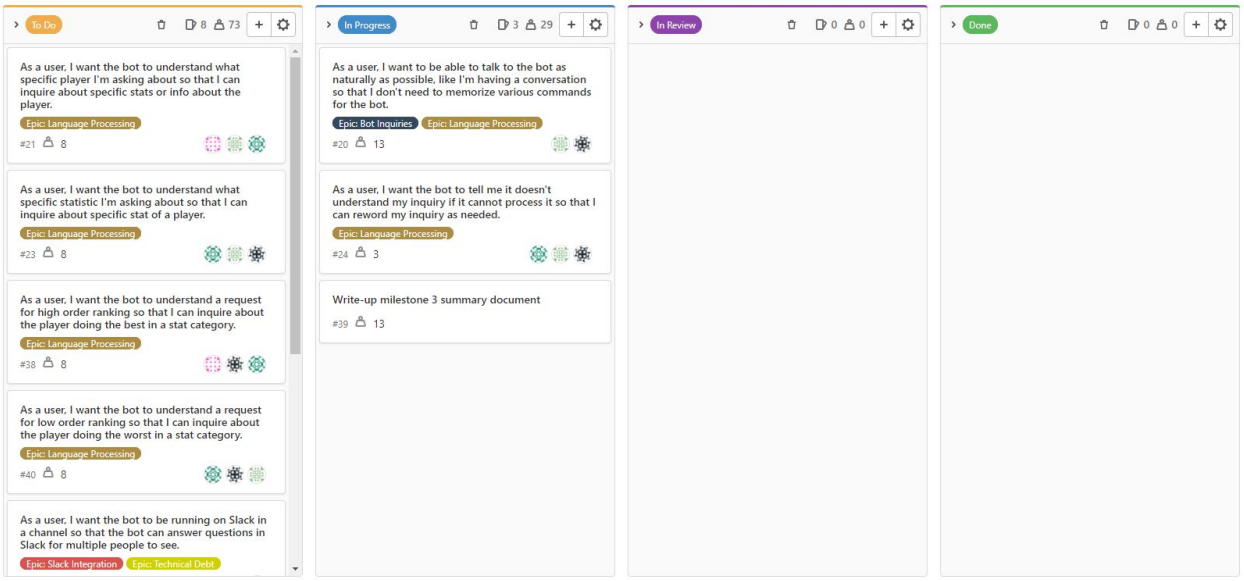
- The code implementation for this user story has been approved by all members of the team - it is clean, refactored, and follows our coding conventions.
  - Code in *nlp.js* and *wit.js* was refactored to improve code readability and ease of maintenance

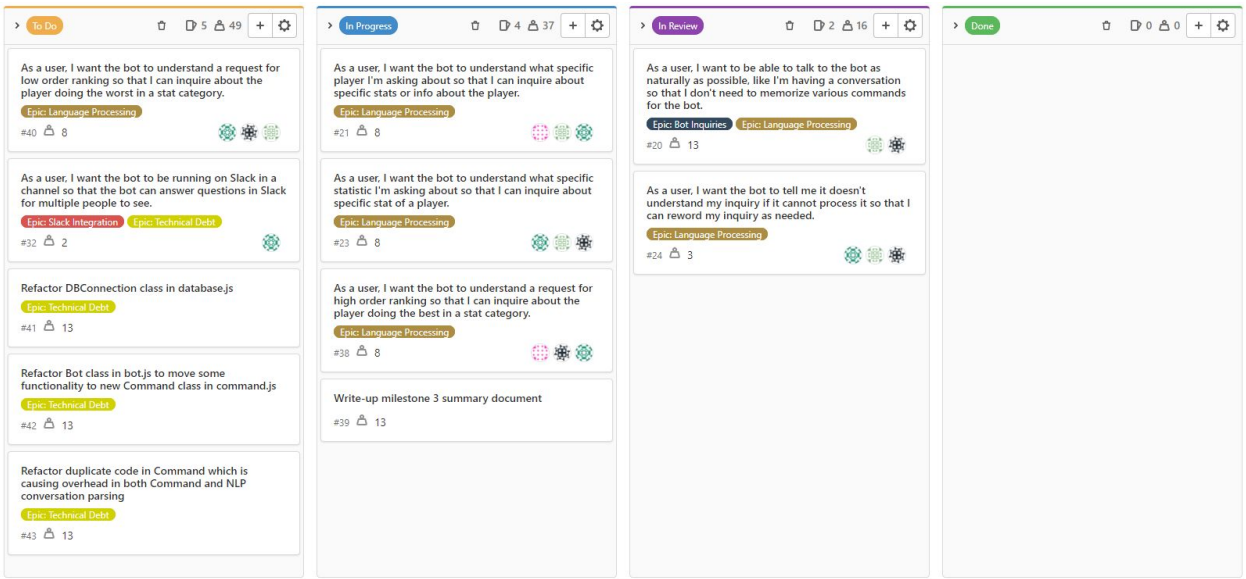
```
retrieveFirstContactEntity(message) {
    return message.entities['wit$contact:contact'][0].value;
    let playerName = message.entities['wit$contact:contact'][0].value;
    // remove 's from the name
    playerName = playerName.replace(/'s/, '');
    playerName = playerName.replace(/'/, '');
    return playerName;
}
```

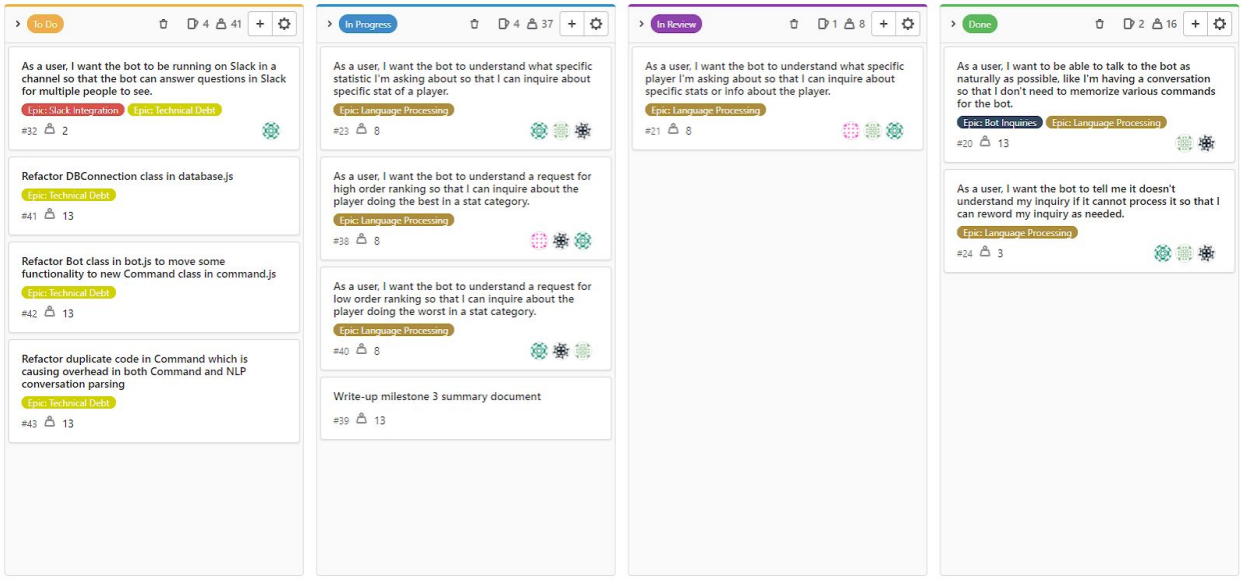
- As can be seen above, logic for manipulating the *playerName* string was moved from *nlp.js* in the top picture, to *wit.js* in the bottom picture. This allowed us to combine functionality for retrieving the player name from wit.ai with functionality for removing *'s* and *'* from the ends of names in user's queries. This resulted in more clean and concise code.
- Additionally, code for retrieving the player name was moved inside of each switch statement for a particular command, rather than being done before the intent is decided.
- All of the changes (https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/merge_requests/12/diffs) follow our coding conventions for using Pascal case when naming files and classes, using camel case when naming variable and functions, having appropriate variable names, and adding descriptive comments when necessary.

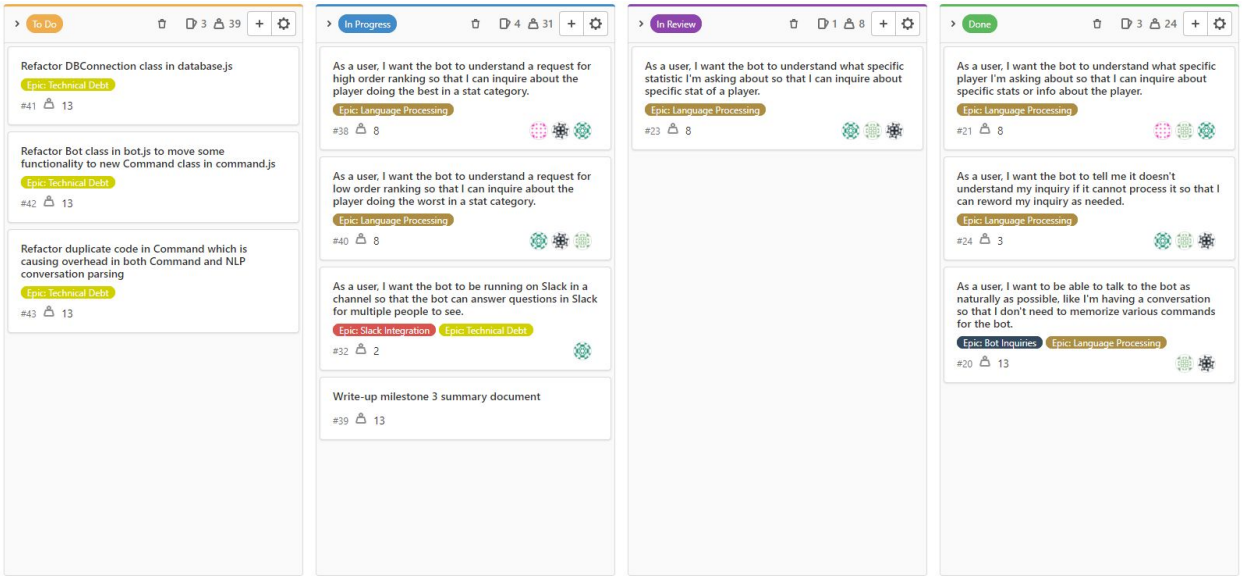
Scrum Board Screenshots of Progress Over Time

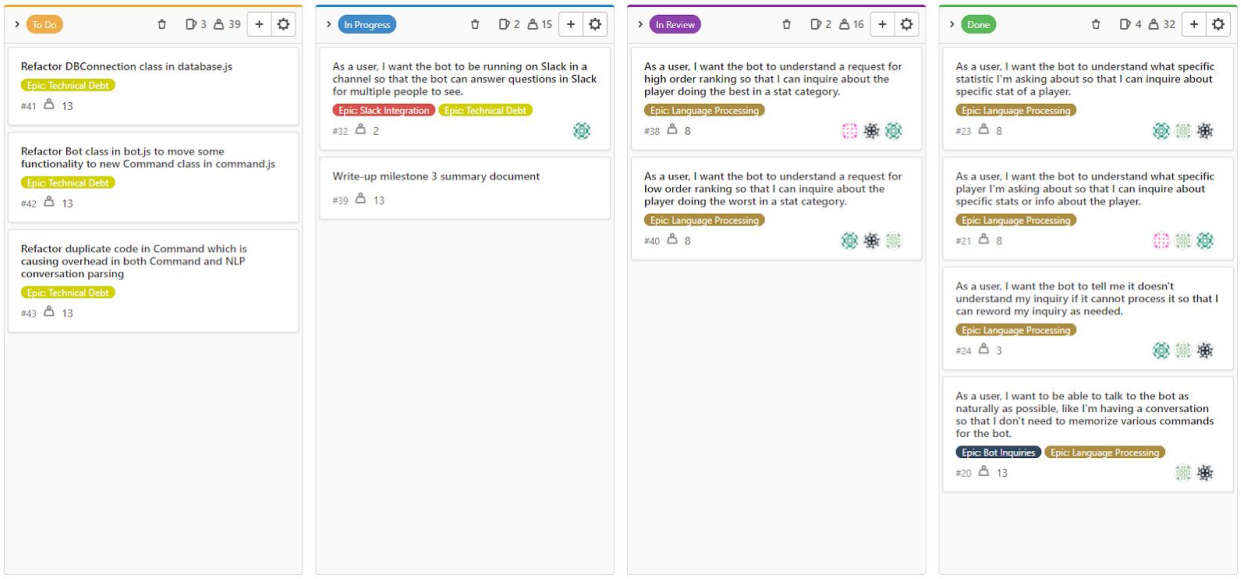
- Top is oldest screenshot, bottom is most recent - taken periodically over the sprint

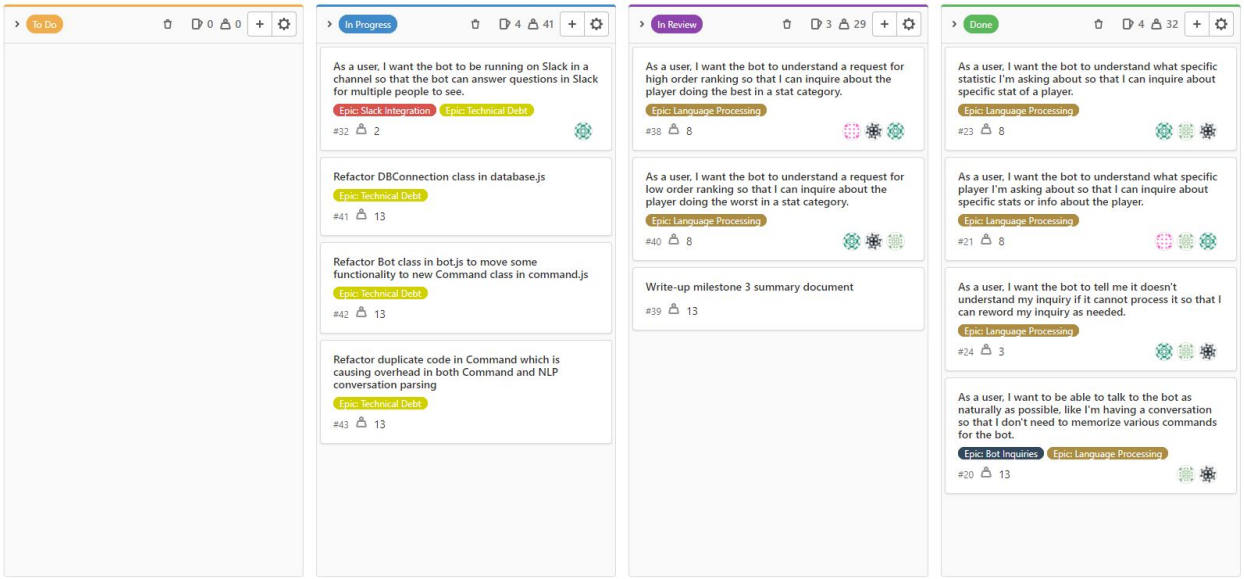


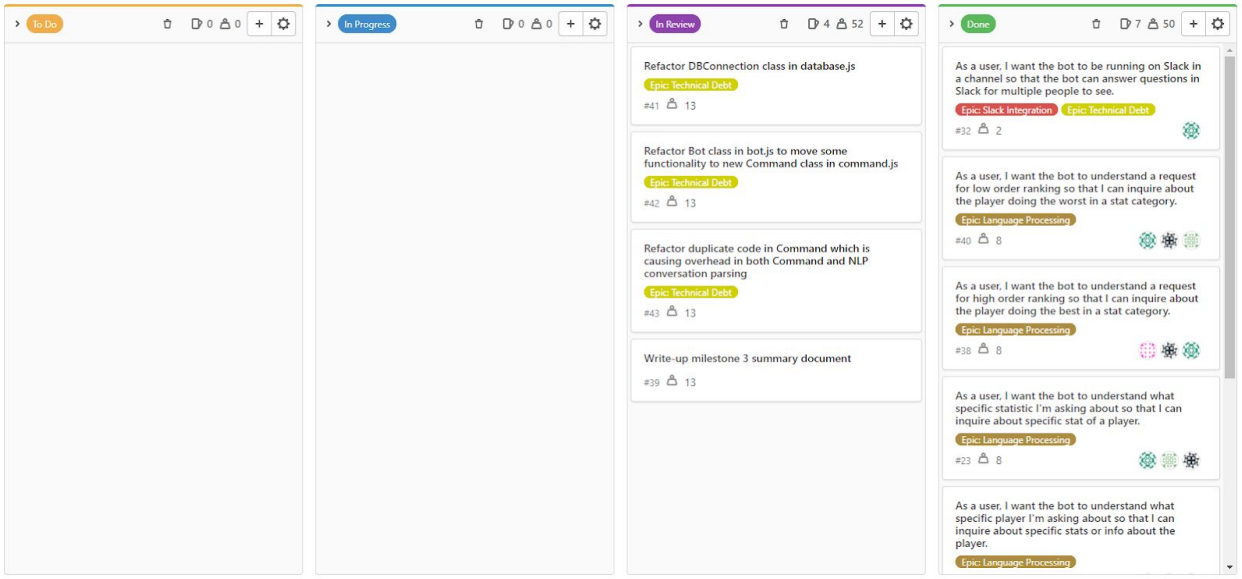


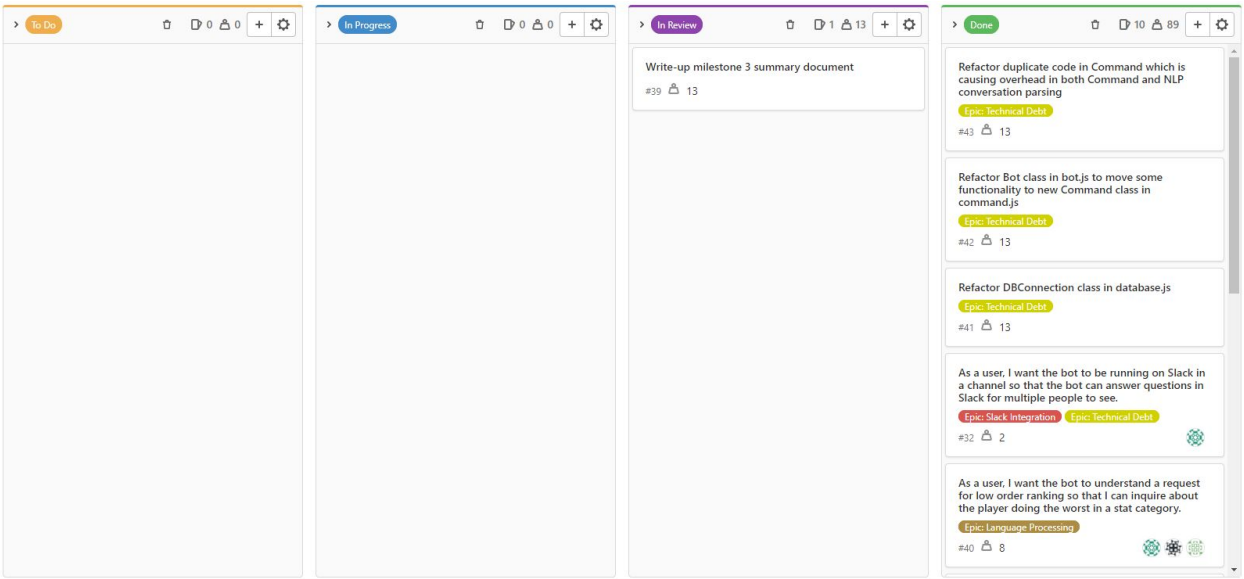

To Do
4 41
As a user, I want the bot to be running on Slack in a channel so that the bot can answer questions in Slack for multiple people to see.
Epic: Slack Integration
Epic: Technical Debt
#32 2
Refactor DBConnection class in database.js
Epic: Technical Debt
#41 13
Refactor Bot class in bot.js to move some functionality to new Command class in command.js
Epic: Technical Debt
#42 13
Refactor duplicate code in Command which is causing overhead in both Command and NLP conversation parsing
Epic: Technical Debt
#43 13
In Progress
4 37
As a user, I want the bot to understand what specific statistic I'm asking about so that I can inquire about specific stat of a player.
Epic: Language Processing
#23 8
As a user, I want the bot to understand a request for high order ranking so that I can inquire about the player doing the best in a stat category.
Epic: Language Processing
#38 8
As a user, I want the bot to understand a request for low order ranking so that I can inquire about the player doing the worst in a stat category.
Epic: Language Processing
#40 8
Write-up milestone 3 summary document
#39 13
In Review
1 8
As a user, I want the bot to understand what specific player I'm asking about so that I can inquire about specific stats or info about the player.
Epic: Language Processing
#21 8
Done
2 16
As a user, I want to be able to talk to the bot as naturally as possible, like I'm having a conversation so that I don't need to memorize various commands for the bot.
Epic: Bot Inquiries
Epic: Language Processing
#20 13
As a user, I want the bot to tell me it doesn't understand my inquiry if it cannot process it so that I can reword my inquiry as needed.
Epic: Language Processing
#24 3


To Do
3 39
Refactor DBConnection class in database.js
Epic: Technical Debt
#41 13
Refactor Bot class in bot.js to move some functionality to new Command class in command.js
Epic: Technical Debt
#42 13
Refactor duplicate code in Command which is causing overhead in both Command and NLP conversation parsing
Epic: Technical Debt
#43 13
In Progress
4 31
As a user, I want the bot to understand a request for high order ranking so that I can inquire about the player doing the best in a stat category.
Epic: Language Processing
#38 8
As a user, I want the bot to understand a request for low order ranking so that I can inquire about the player doing the worst in a stat category.
Epic: Language Processing
#40 8
As a user, I want the bot to be running on Slack in a channel so that the bot can answer questions in Slack for multiple people to see.
Epic: Slack Integration
Epic: Technical Debt
#32 2
Write-up milestone 3 summary document
#39 13
In Review
1 8
As a user, I want the bot to understand what specific statistic I'm asking about so that I can inquire about specific stat of a player.
Epic: Language Processing
#23 8
Done
3 24
As a user, I want the bot to understand what specific player I'm asking about so that I can inquire about specific stats or info about the player.
Epic: Language Processing
#21 8
As a user, I want the bot to tell me it doesn't understand my inquiry if it cannot process it so that I can reword my inquiry as needed.
Epic: Language Processing
#24 3
As a user, I want to be able to talk to the bot as naturally as possible, like I'm having a conversation so that I don't need to memorize various commands for the bot.
Epic: Bot Inquiries
Epic: Language Processing
#20 13


To Do
3 39
Refactor DBConnection class in database.js
Epic: Technical Debt
#41 13
Refactor Bot class in bot.js to move some functionality to new Command class in command.js
Epic: Technical Debt
#42 13
Refactor duplicate code in Command which is causing overhead in both Command and NLP conversation parsing
Epic: Technical Debt
#43 13
In Progress
2 15
As a user, I want the bot to be running on Slack in a channel so that the bot can answer questions in Slack for multiple people to see.
Epic: Slack Integration
Epic: Technical Debt
#32 2
Write-up milestone 3 summary document
#39 13
In Review
2 16
As a user, I want the bot to understand a request for high order ranking so that I can inquire about the player doing the best in a stat category.
Epic: Language Processing
#38 8
As a user, I want the bot to understand a request for low order ranking so that I can inquire about the player doing the worst in a stat category.
Epic: Language Processing
#40 8
Done
4 32
As a user, I want the bot to understand what specific statistic I'm asking about so that I can inquire about specific stat of a player.
Epic: Language Processing
#23 8
As a user, I want the bot to understand what specific player I'm asking about so that I can inquire about specific stats or info about the player.
Epic: Language Processing
#21 8
As a user, I want the bot to tell me it doesn't understand my inquiry if it cannot process it so that I can reword my inquiry as needed.
Epic: Language Processing
#24 3
As a user, I want to be able to talk to the bot as naturally as possible, like I'm having a conversation so that I don't need to memorize various commands for the bot.
Epic: Bot Inquiries
Epic: Language Processing
#20 13


To Do
0 0
In Progress
4 41
As a user, I want the bot to be running on Slack in a channel so that the bot can answer questions in Slack for multiple people to see.
Epic: Slack Integration
Epic: Technical Debt
#32 2
Refactor DBConnection class in database.js
Epic: Technical Debt
#41 13
Refactor Bot class in bot.js to move some functionality to new Command class in command.js
Epic: Technical Debt
#42 13
Refactor duplicate code in Command which is causing overhead in both Command and NLP conversation parsing
Epic: Technical Debt
#43 13
In Review
3 29
As a user, I want the bot to understand a request for high order ranking so that I can inquire about the player doing the best in a stat category.
Epic: Language Processing
#38 8
As a user, I want the bot to understand a request for low order ranking so that I can inquire about the player doing the worst in a stat category.
Epic: Language Processing
#40 8
Write-up milestone 3 summary document
#39 13
Done
4 32
As a user, I want the bot to understand what specific statistic I'm asking about so that I can inquire about specific stat of a player.
Epic: Language Processing
#23 8
As a user, I want the bot to understand what specific player I'm asking about so that I can inquire about specific stats or info about the player.
Epic: Language Processing
#21 8
As a user, I want the bot to tell me it doesn't understand my inquiry if it cannot process it so that I can reword my inquiry as needed.
Epic: Language Processing
#24 3
As a user, I want to be able to talk to the bot as naturally as possible, like I'm having a conversation so that I don't need to memorize various commands for the bot.
Epic: Bot Inquiries
Epic: Language Processing
#20 13





Note: of course, the write-up summary task is “Done” when this is submitted.

Optional Elements

Note for Prof. Judi McCuaig: see team standards wiki document for explanation: https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/wikis/TeamStandards#m2-correctional-explanation-for-devops-branchmerge-strategy

1. Code Smells

1.1 Naming Smells

Definition

- A naming smell is a code smell that comes from poor naming choices for things like variables and functions. This includes excessively long, short, and/or vague names.
- This may indicate a deeper-level issue where the purpose of the variable or function is not obvious and not clearly defined.

Reasons Why These Code Smells Are Common to This Type of Project

- Naming smells are common to every type of coding project, including this one. Developers need to decide what to name their variables and functions all the time. If their purpose isn't clear-cut and defined, it's usually reflected in a poor naming choice.

Example (refactoring, links to the code base and tickets)

- A code smell was identified for a variable name "key" (bad naming choice) and suggested a small refactor to rename it to something more descriptive ("stat"). In merge request 9 (https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/merge_requests/9), Seegal

fixed the code smell in *command.js* by implementing the suggestion (among other refactors).

- Previous code with the code smell (variable ‘key’ is not a descriptive name):

  https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/blob/56fb9ff0e5ede02bb8978b37dbf07ff0210e5613/bot.js

```
// look at the command entered, and get the stat desired
switch (command) {
case '/player': { // USAGE: /player <playerName>, RETURNS all stats of given player
    res.shift();
    const playerName = res.join(' ');
    const response = await player.getPlayerCommand('s19', playerName);
    if (typeof response === 'string') {
        await context.sendActivity(MessageFactory.text(response, response));
    } else {
        for (var key in response[0]) {
            const msg = key + ': ' + (response[0])[key];
            await context.sendActivity(MessageFactory.text(msg, msg));
        }
    }
    break;
}
```

- As you can see in the below image, it now checks **stat** in response, rather than **key** in response:

  https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/blob/e2c58e6bfd787b00c4d3

083d0b950651d9341be4/impl/command.js

```
async getPlayerCommand(playerName) {
    return new Promise((resolve, reject) => {
        const player = new Player();
        player.getPlayerStats('s19', playerName)
            .then((response) => {
                if (typeof response === 'string') return resolve(response);
                else {
                    let msg = '';
                    for (var stat in response[0]) {
                        msg += `${ stat } : ${ (response[0])[stat] }  \n`;
                    }
                    return resolve(msg);
                }
            })
            .catch((error) => resolve(error.message));
    });
}
```

- The variable ‘key’ was changed to ‘stat’ to more clearly reflect what it’s being used for. In this case, it did not indicate the underlying problem of being poorly defined.

1.2 Large Function/Method

Definition

- A large function/method is a code smell that comes from excessively large functions.
- This may indicate a deeper-level issue where the responsibility of the function is too broad (i.e. it does too many things). This makes it difficult to read and maintain.
- This can be mitigated by splitting the function into multiple smaller ones.

Reasons Why These Code Smells Are Common to This Type of Project

- Our project makes use of Object Oriented Programming (OOP). When developing classes, it is common to make the mistake of stuffing multiple functionalities/jobs into one method, resulting in it becoming excessively large.

Example (refactoring, links to the code base and tickets)

- We noticed that we were placing all the command handling functionality in a single function, which made it grow much larger with each command we added. This was identified as the 'large function/method' code smell. In merge request 9 (https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/merge_requests/9), Seegal abstracted the command parsing functionality out of *bot.js* and created a separate class in *command.js* called 'Command'. There, each command had its own small function to handle the command.
- Too much code to post screenshots here. See the following link and observe the changes in *bot.js* and *command.js*: https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/merge_requests/9/diffs.
    - Notice how a large amount of code was removed from the async arrow function in bot.js and replaced in commands.js. Also notice how some functionality in each case in the large switch statement has been extracted out into smaller methods at the bottom of *command.js*.

1.3 Duplicate Code

Definition

- Duplicate code is a code smell that comes from copying code that contains desired functionality from one place to another.
- This indicates that the functionality needs to be abstracted and/or generalized into a common place that multiple clients can access as needed.
- This increases complexity for maintenance as modifications to one copy also likely need to be made in other copies. This also increases risk because duplicating poor and/or buggy code will introduce the same problems in multiple locations within the code base.

Reasons Why These Code Smells Are Common to This Type of Project

- It is common for developers to become lazy when developing a feature and copying code from somewhere else that has the functionality they need.
- It may also be the case where code is duplicated because the developer doesn't have enough understanding of either the programming language itself or how the duplicated code works, so they are afraid to modify it.
- Also, sometimes, by no fault of the developers themselves, the project may just be tight on time, and they do not have time to properly abstract common functionality, so code is not duplicated.

Example (refactoring, links to the code base and tickets)

- We identified duplicate code in the code for /playerLowest and /playerHighest. In merge request 9 (https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/merge_requests/9), Seegal addressed this by combining the two into a single function.
- Duplicate code: https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/blob/fba2ec58172581b0ce98ef4d0e3632b86ed63c80/impl/player.js

```
async getHighestCommand(table, stat) {
    return new Promise((resolve, reject) => {
        const db = new DBConnection();
        const query = 'SELECT PLAYER_NAME, ' + stat + ' FROM ' + table + ' ORDER BY ' + stat + ' DESC;';
        db.query(query)
            .then((response) => resolve(response))
            .catch(() => reject(Error('Stat passed is invalid.')));
    });
}
```

```
async getLowestCommand(table, stat) {
    return new Promise((resolve, reject) => {
        const db = new DBConnection();
        const query = 'SELECT PLAYER_NAME, ' + stat + ' FROM ' + table + ' ORDER BY ' + stat + ' ASC;';
        db.query(query)
            .then((response) => resolve(response))
            .catch(() => reject(Error('Stat passed is invalid.')));
    });
}
```

- Refactored code:

  https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/blob/e2c58e6bfd787b00c4d3083d0b950651d9341be4/impl/player.js

```
async getStatHighestLowestPlayer(table, stat, order) {
    return new Promise((resolve, reject) => {
        const db = new DBConnection();
        const query = `SELECT PLAYER_NAME, ${ stat } FROM ${ table } ORDER BY ${ stat } ${ order };`;
        db.query(query)
            .then((response) => resolve(response))
            .catch(() => reject(Error('Stat passed is invalid.')));
    });
}
```

2. Refactoring

2.1 Refactoring the DBConnection Class

- In issue #41 (https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/issues/41), we identified an improvement that could be made inside of the “DBConnection” class, in *database.js.*
- **The Problem**
    - The “DBConnection” class involved multiple functions for the supported commands and was used to establish a connection to the database and execute a queries for functions.
    - Having separate functions in this class for each specific command resulted in duplicate code being used for the connections to the server.
- **The Solution**
    - We decided to refactor the “DBConnection” class to remove duplicate code.
    - The “DBConnection” class was reduced to a single function for establishing a connection to the database, and executing a query passed as a parameter.

```
async query(query) {
    return new Promise((resolve, reject) => {
        const connection = sql.createConnection({
            host: 'chatron.socs.uoguelph.ca',
            user: 'sysadmin',
            password: 'SublimeVarnish',
            database: 'NBA'
        });

        connection.query(query, (err, result, fields) => {
            connection.end(); // close connection after call
            if (err) return reject(err);
            return resolve(JSON.parse(JSON.stringify(result)));
        });
    });
}
```

    - The other functions in the “DBConnection” class were removed, and instead queries were passed into the function as parameters from *player.js.*

- For example, instead of having a function in the “DBConnection” class called *findPlayerSingleStat()*, which would execute a specific query, the query passed as a parameter from the function *getPlayerCommand()* in *player.js.*

```
db.findPlayerSingleStat(table, playerName, stat).then((response) => {
const query = 'SELECT ' + stat + ' FROM ' + table + " WHERE PLAYER_NAME='" + playerName + "';";
db.query(query).then((response) => {
```

- The full list of changes can be seen in the following commit: https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/commit/1a3907a6c0944879b13cb6331d7b37051446acd5

- As a result of the refactor, duplicate code was removed and it was easier to write and maintain code for connecting to the database and using commands. When querying the database, we no longer had to create a new function in *database.js* specifying the query to be executed. Instead, new commands could quickly be written outside of *database.js*, such as this example from the full list of changes:

```
const query = 'SELECT * FROM ' + table + " WHERE PLAYER_NAME='" + playerName + "';";
db.query(query).then((response) => {
    if (response.length > 0) resolve(response);
    else resolve('Player ' + playerName + ' not found.');
}).catch((error) => {
    reject(error);
});
```

2.2 Refactoring the Bot Class

- In issue #42 (https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/issues/42), we identified an improvement that could be made inside of the “Bot” class, in *bot.js.*
- **The Problem**
    - All of the command handling functionality was in a single function; however, as we added more supported commands, the function grew larger and more difficult to maintain.

- The parsing capability being added to *bot.js* blurred the purpose of the *bot.js* class by giving it multiple roles: 1. *handling the context of sending/receiving messages*; and 2. *parsing of the message itself.*
    - Simply put, the class was “doing too many things” and by making sure *bot.js* focused on routing the message, we could move all non-essential code to another class that focused on parsing the message.

- **The Solution**
    - We decided to create a separate “Command” class in *command.js*, and later the “NLP” class in *nlp.js*. The bot class would immediately hand off the message received to these classes, and send the response generated.
    - The “Command” class involved a similar switch statement for recognizing commands as was in *bot.js*, but with the logic of each command extrapolated into separate functions.
    - For example, before the refactor, the functionality for */playerSingleStat* was in the switch statement in *bot.js*:

```
case '/playerSingleStat': { // USAGE: /playerSingleStat <stat> <playerName>, RETURNS stat requested
    const stat = res[1];
    res.shift();
    res.shift();
    const playerName = res.join(' ');
    try {
        const response = await player.getPlayerSingleStatCommand('s19', playerName, stat);
        if (typeof response === 'string') {
            await context.sendActivity(MessageFactory.text(response, response));
        } else {
            const msg = 'Player \'' + playerName + '\' ' + stat + ' is ' + (response[0])[stat];
            await context.sendActivity(MessageFactory.text(msg, msg));
        }
    } catch (error) {
        await context.sendActivity(MessageFactory.text(error.message));
    }
    break;
}
```

    - Following the refactor, a similar switch statement in *command.js* would instead call a separate function called *getPlayerSingleStatCommand()*.

```
case '/playerSingleStat': { // USAGE: /playerSingleStat <stat> <playerName>, RETURNS stat requested
    const stat = res[1];
    res.shift();
    res.shift();
    const playerName = res.join(' ');
    this.getPlayerSingleStatCommand(playerName, stat)
        .then((response) => resolve(response))
        .catch((error) => reject(error));
    return;
}
```

```
async getPlayerSingleStatCommand(playerName, stat) {
    return new Promise((resolve, reject) => {
        const player = new Player();
        player.getPlayerSpecificStat('s19', playerName, stat)
            .then((response) => {
                if (typeof response === 'string') return resolve(response);
                else return resolve(`Player ${ playerName } has ${ response[0][stat] } ${ stat.toUpperCase() }`);
            })
            .catch((error) => reject(error));
    });
}
```

    - The full list of changes can be seen in the following commit: https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/commit/6d1a8150d3448a4e10e1b31f6d1cffa37ece5f89

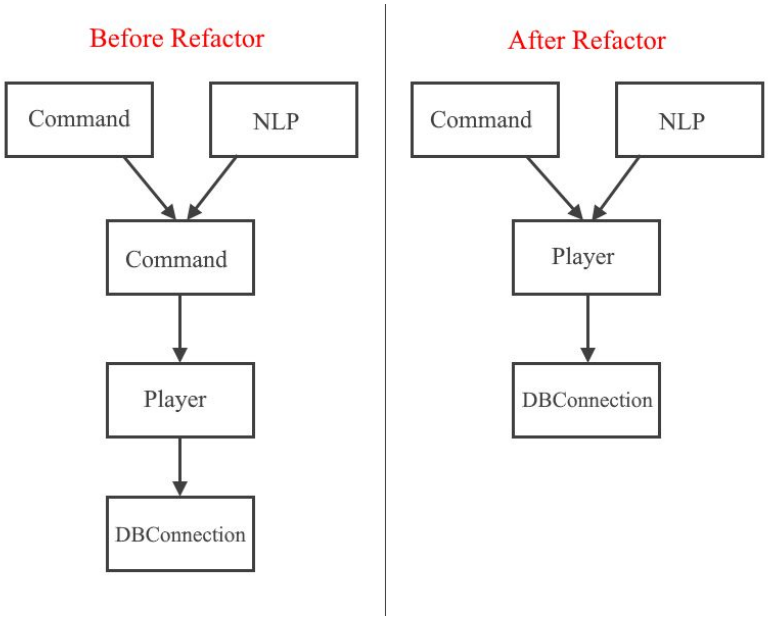
- As a result of the refactor, the code was easier to read and maintain while providing easier ways for the team to add support for additional commands. Only a new switch statement was necessary to add, and all the logic for the command could be contained in a separate function in *command.js*.
- Additionally, by having each class constrained to its own purpose, we greatly improved code readability.

2.3 Refactoring the Command Class and Player Class

- In issue #43 (https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/issues/43), we identified an improvement that could be made inside of the “Command” and “Player” classes (*command.js* and *player.js*, respectively).
- **The Problem**

- Both of the “Command” and “NPL” classes are meant to parse user input passed to them from *bot.js*, but there is duplicate code in *command.js* which will essentially take the parsed input, do nothing with it, then pass it along to *player.js*.
- This unnecessary overhead also affects *nlp.js*, which passes the parsed *name* and *stat* to these functions which do not affect the modification of the message.
- Additionally, the “Player" class only served to send a query to the database and then pass the response back to *command.js*.

- **The Solution**
  - We decided it would be beneficial for the messages to “skip” the methods in the Command class, and instead go directly to *player.js* where it will achieve the same functionality without an unnecessary asynchronous function in between.
  - The following is a visual to help illustrate the flow before and after:



  - Before: The parsed *name* and *stat* from *command.js* and *nlp.js* would go to functions inside *command.js*, which were duplicated to Player. Yes, *command.js* basically sent this data to other functions within the same file.

    - After: The parsed *name* and *stat* skip the duplicate functions in Command (which were removed) and go directly to *player.js*, which handles the response and error better.
    - Additionally, the “Player” class was refactored so that errors are handled within that class, rather than just passing the response from the database to *command.js*.
    - The full list of changes can be seen in the following commit: https://git.socs.uoguelph.ca/3760f20/chatron/wiki-chatbot/-/commit/ab8df0bbc268e18f0ecacb7e8dc95b52906c8083
- As a result of the refactor, unnecessary and duplicate code was removed. Errors are handled in a more robust way in *player.js* and code for handling commands and natural language processing in *command.js* and *nlp.js* is easier to maintain and add additional functionality to in the future. Most importantly, the logical flow is better now.