Group Project Requirements Specification

Milestone 3

Many students just briefly scan this document, this inevitably leads to missed requirements and lost grades. Don't let this happen to you, or your group! ***Read this document carefully and ask questions if you have any concerns or questions. Assumptions are dangerous, do not make them, clarify!***

Disclaimer

I will continue to make mistakes from time to time, and so *I continue to reserve the right to alter this document and inform you of changes when I need to correct those mistakes. I will always endeavour to do so in a fair way.*

Progress so far

You have continued to use test-driven development to develop your code, so it comes well tested. You all now have a working continuous integration system that monitors the health of your systems.

Groups in our course have ended up with different numbers of group members. For this reason your project may have diverged from the functionality of other groups. No need to worry, this milestone's epics will work in all of our group configurations.

We continue to develop our hockey simulation system.

It's time for change

This milestone you are going to do some big changes to your existing code. This will give you practice in experiencing the sudden impact of big changes. If you have prepared for these changes by following the best practices taught in this course the changes will be reasonably simple to implement. If you have not, you will see that these changes are wide sweeping across your whole project and difficult to implement. Everyone will be expected to participate in these changes to ensure you all experience either the joy or pain of this process. ;)

It's time to trust your own abilities

The requirements below are often very loosely defined. I trust you to make your own smart decisions. It's time to see you design something that accomplishes the intended goals of the epic, without having to be handed every detail.

Milestone 3 Requirements

General requirements for the whole team:

- All code you write in this milestone, except for the classes below, will be in the **business layer** (to do with the business of simulating a season of hockey) and therefore must be implemented using test-driven development (TDD) such that they are 100% covered by unit tests.
    - **Exceptions:**
        - Small input/output classes in your **presentation layer** that deal with communicating with the user of the simulation system via the command line.
        - Small persistence classes in your **data layer** that implement a persistence interface for **business layer** classes to persist themselves.
    - **Remember, you must write your own tests.** The easiest way to ensure this is to actually use test-driven development and write your tests first. Done properly, this way of working guarantees 100% on TDD for you because code won't exist without being covered by a test!
- Continue to monitor the health of your system and follow best practices:
    - **Use Gitflow**. Do not recklessly commit to master, we need master clean to evaluate your code this milestone.
    - **Maintain your CI/CD system** and keep your tests passing.
    - **Don't leave things to the last minute.** Work at a steady pace, 10 hours per week throughout the milestone so that your fellow group members can develop their work alongside you. Though you are marked individually for the most part you are not working in a vacuum, other group members will depend on the classes you write. Be a good group member. This milestone we will add a professionalism element to the marking rubric to encourage you to maintain a steady pace (Principle 8 of Agile methodology).
- Any modifications to the JSON import should be validated as appropriate, continue to ensure the JSON you are importing is valid both syntactically and for the data values themselves.

Your TA will assign each of you to one of the following MAJOR epics. In addition, you will each participate in the implementation of the group work epic. You are responsible for all tasks in your individually assigned epic as well as the tasks assigned from your shared group work epic.

- **Group Work Epic: (All group members share tasks equally)**
    - Epic goal: To give students experience handling change, and to experience the benefits of creational patterns. All students in the group should take at least some part in the subtasks in this epic, making changes to the entire codebase. Do all of this work **first**, before beginning the other epics.

  - Switch to the State pattern. We're doing state machines in our program, now that we know about design patterns we must use them. If your team did not use the State pattern from the beginning, switch to it now. It is your team's choice where to perform state transitions (in the state machine loop, or have the states return the states to switch to).
  - Switch all instantiation in your project to use creational patterns. Refactor your existing code, and use them in your new code. Ideally, get your decision point for the concrete pattern choices to a single point in your system, for example your main().
  - Implement error handling and logging. Add logging to all major events in the system and business logic decision making. Ensure your error handling logic is following best practices. Implement a global exception handler that logs uncaught exceptions.
  - **Switch all database classes you've written to JSON serialization / deserialization.** Right now all of your objects in your league model should be persisting themselves through a database class that implements an interface, passed to the object. Use the serialization logic built for your system in the previous milestone and implement deserialization to convert your system to using JSON to store your league between executions. When requested by the state machine, persist to one or more JSON files in a folder. When loading a team, use the available files in the folder to determine which teams can be loaded. This task will either be easy or hard, depending on the work you've done thus far.
  - Grow the number of players on a team to 30. Distinguish between an "active" roster and an "inactive" roster. The active roster of 20 players are the 20 best players (18 skaters, 2 goalies), the inactive roster of 10 players are injured players or reserve players that hang around in case a better player becomes injured. When players are injured, swap them for an uninjured player on the inactive roster (if there is one). When players recover from injuries, if they are better than players on the active roster swap them back to the active roster. Training and trading affects all players (inactive or active).
- **Player Draft:**
  - Epic goal: To replenish teams and the free agent list by generating new players to feed into the simulation system each year.
  - First, give players a birthday! (See the new JSON). Remove the age attribute from JSON import and switch the age property of your Playe class to be calculated. Modify the existing aging logic to use the birthday to determine when the player's age increments.
  - Use the new statDecayChance field in the aging section of gamePlayConfig to determine on a player's birthday whether their stats decay due to age. For each statistic of a player on their birthday use the statDecayChance to determine whether that statistic decreases by one point.
  - Implement the player draft (see Appendix C – Draft Rules for details).

    - Create the drafting state, insert it into the state machine such that the state is entered on July 15th. This state performs the draft and resolves team rosters.
    - Once the draft is complete every team must drop down to 30 players. This process is complicated in the real world and involves a complex system of minor leagues and free agency. In our system the team will simply pick their 30 best players, including the new draftees. The final team of 30 players must have 16 forwards, 10 defense, 4 goalies. Excess players are released to free agency. Ignore injuries.
    - Work with the programmer that will be improving trading to provide a mechanism for teams to trade their future draft picks. You must record when teams have traded away their picks, or gained picks and incorporate this into the logic when performing the rounds of the draft. **Do this work early so that the trading player has stubs or interfaces to program to.**
- **Game Simulation:**
    - Epic goal: To improve the realism of simulated games. Your team leads want you to increase the realism of the game simulation step. They are rendering and animation programmers though so they don't know much about AI. They haven't given you any ideas on how to approach this problem. The only feedback they've given is "Hockey is a game about star players, the random win chance and the generic team strength calculation result in games that are won more than they should be and a reduced feeling of success when you successfully trade to get a team stacked full of star players. We suggest you take a 'shots on goal' approach." Assign this epic to your team member that knows the most about hockey, or at least is the most interested in learning more about hockey.
    - Add a section in your app's configuration for configuring tuning values for your game simulation algorithm.
        - Add a value named penaltyChance, you will use this to tune your system to match the NHL penalty average.
        - You may need to add other values that you use to tune your algorithm, do so at your own discretion.
    - Write an algorithm that calculates the winner of a game based on the advice given in Appendix D – Game Simulation Notes.
    - At the end of the regular season, output the league averages for the following data:
        - Goals per game
        - Penalties per game
        - Shots
        - Saves
- **Improve Trading:**
    - Epic goal: To make trading a bit more realistic and to resolve some of the silliness of the system.
    - Add personalities to the general manager. Note the new JSON definition which adds the following:
        - GM's now have a name and a personality field.

        - The trading section of the gameConfig now has a gmTable field that lists possible GM personality values and corresponding modifiers to the randomAcceptanceChance. Apply these modifiers to the random acceptance chance logic. Note there does not appear to be any limit to the number of personality types possible. The gmTable field appears to be a dictionary.
    - Add draft picks to the trading system. A team can trade away their future draft picks in exchange for players:
        - Every draft has 7 rounds. A team does not know what their pick order will be in the future draft, but they can trade their pick from each of the 7 rounds, and this pick generally is considered to have significant value:
            - First round draft picks are equivalent to trading a star player
            - Seventh round draft picks are slightly better than trading an average player
        - Track which teams have traded their picks to which other teams. The draft system will need to either be passed this information, or able to query it from the trading system. You must decide where this responsibility should lie based on your group's designs.
    - Resolve deficiencies in previous trading implementations:
        - Find a way to allow a team to trade one player for multiple players, or multiple players for one player.
        - Find a way to allow trading players of different positions. Find a way to calculate their relative strength. The best goalie, defense or forward player should all hold equivalent value to the AI decision making logic.
        - Find a way to resolve all problems with the team's rosters after a trade is completed. If players need to be dropped, the team drops its weakest players to the free agent list while ensuring the rosters remain valid (16 forwards, 10 defense, 4 goalies). If players need to be added, the team hires the best players of needed positions from the free agent list.
- **Trophy System: (Omit for groups of 3 or less)**
    - Epic goal: To begin the work of simulating the various trophies and awards given out each season in the course of a hockey league (specifically in our case the NHL).
    - Review Appendix E – Trophies for a list of trophies to be awarded by this system.
    - Use the **Observer** behavioural design pattern to implement a system(s?) capable of tracking the various events necessary to award the trophies defined in Appendix E.
    - Create a Trophy state and insert it into the season simulation state machine after the Stanley Cup winner is determined, and before advancing to the next season.
        - Write a generic algorithm that uses data collected by the Observer system implemented above to calculate and award the various trophies.
        - Maintain a historical record of the trophy winners throughout the entire history of the league.

        - Each year, output the historical (and new) winners of each award, ordered by most recent to oldest.
- **All-Star Game: (Omit for groups of 4 or less)**
    - Epic goal: To create a smaller epic so that the 5th programmer can assist on the larger amount of code that needs to be refactored in the group epic. We will generate the teams for, simulate and output the results of the All-Star game.
    - First, this programmer should do at least **half** of the work in the group epic. This is to lighten the load since the amount of code is greater in this group.
    - Implement the All-Star Game:
        - The All-Star game is a special game held on the last Saturday of the month of January, the natural half-way point through the season.
        - Two teams formed of the best players from each conference face off!
            - Create the All-Star game state and insert it into the season simulation state machine to ensure it is triggered on the last Saturday of January. Modify the existing scheduling algorithm to ensure no other games are scheduled on this day.
            - Generate the two teams by analyzing all other teams in each conference. Free Agents are not eligible for the all-star game.
                - Figure out how to ensure each team in the conference must be represented by at least 1 player on their conference's team.
                - Pick the 20 best players from each side to form the teams.
            - Use the game simulation logic to determine the result of the All-Star game.
            - Record the results (participants from each conference, and the team they were playing on, the score, and winning conference) for the All-Star game for the entire history of the league.
            - When the trophies are announced, include an announcement of the details of the All-Star game for the current season and previous season.

Additional System / Process Requirements

In addition to the functional requirements above your group project must always adhere to the following system and process requirements:

- Your project is built with Java
- You do not include 3rd party packages/components without permission from the instructor. Tag your instructor in your Teams channel with a request to use a technology and wait for a response before using it.

Marking Rubric

Professionalism Grade Cap
The grade you can achieve **as an individual** is capped by the level of professionalism you demonstrate in your group. To earn 100% in this milestone you must deliver all of the tasks you are assigned at a steady pace (some work done each week). Leaving tasks incomplete or leaving the work until the last minute results in a reduced maximum grade for your work.

Grade Cap	Criteria
100	Completed **all tasks** at a reasonable pace as determined by your TA (33% each of the 3 weeks of the milestone).
90	Completed all tasks by the milestone deadline (11:59PM!)
70	Did not complete all tasks.
50	Completed 50% or less of assigned tasks.

Monitoring the health of your system, and maintaining process best-practices (10%)

- **CI/CD: 5%**
    - Your team has a working CI/CD system throughout the milestone. Your TA will evaluate this periodically. If they click "Run pipeline" and the pipeline fails you will lose points.
- **Gitflow: 5%**
    - Your team adheres to the Gitflow git workflow.

Assessment (90%)
The remaining 90% of your milestone grade is determined by the results of an assessment of your code by your TA.

This means you must focus on learning, understanding and applying the best practices taught to you before and during this milestone.

Make sure your classes do not violate S.O.L.I.D. principles. Use test-driven development. Evaluate and adjust your packaging to ensure your packages adhere to the principles of cohesion and coupling. Ensure that your code does not violate any layer boundaries or clean code practices. Implement logging and follow error handling best practices. Finally, analyze your code and determine the appropriate use for design patterns. Almost all instantiation in your program should be through creational patterns, also seek out opportunities to correctly use structural and behavioural design patterns.

Remember, you can use the assessment spreadsheet as a check to ensure you are submitting quality work! Assess yourself! Since you will need to do this, you might as well practice.

Appendix A – JSON Import Data

The following JSON defines the structure of the import file:

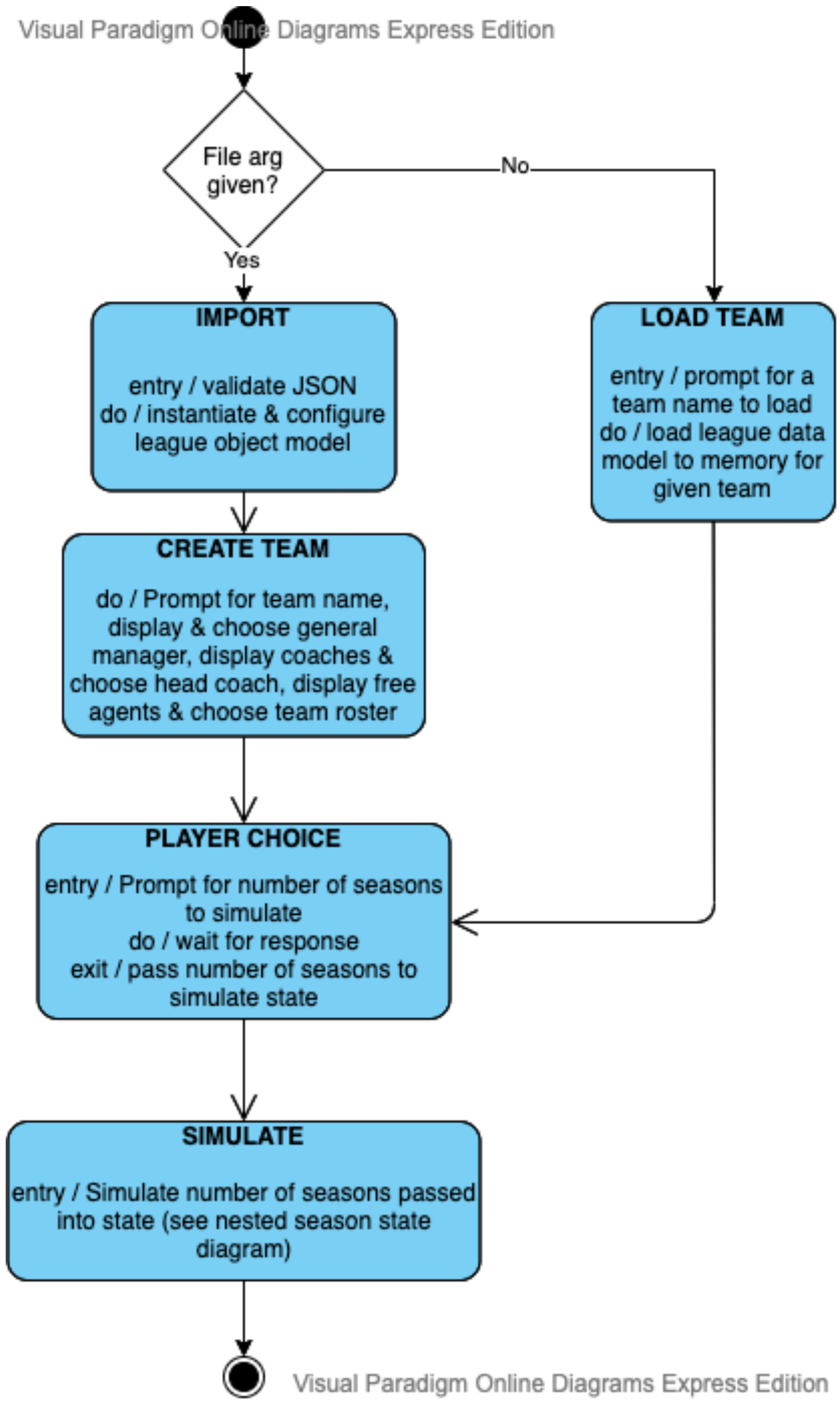
```
{
  "leagueName":"Dalhousie Hockey League",
  "gameplayConfig":{
    "aging":{
      "averageRetirementAge":35,
      "maximumAge":50,
      "statDecayChance":0.05
    },
    "injuries":{
      "randomInjuryChance":0.05,
      "injuryDaysLow":1,
      "injuryDaysHigh":260
    },
    "training":{
      "daysUntilStatIncreaseCheck":100
    },
    "trading":{
      "lossPoint":8,
      "randomTradeOfferChance":0.05,
      "maxPlayersPerTrade":2,
      "randomAcceptanceChance":0.05,
      "gmTable":{
        "shrewd":-0.1,
        "gambler":0.1,
        "normal":0.0
      }
    }
  },
  "conferences":[
    {
      "conferenceName":"Eastern Conference",
      "divisions":[
        {
          "divisionName":"Atlantic",
          "teams":[
            {
              "teamName":"Boston",
              "generalManager":{
                "name":"Bonnie Chang",
  "personality":"normal"
},
"headCoach":{
  "name":"Mary Smith",
  "skating":0.5,
  "shooting":0.8,
  "checking":0.3,
  "saving":0.5
},
"players":[
  {
    "playerName":"Player One",
    "position":"forward",
    "captain":true,
    "skating":15,
    "shooting":18,
    "checking":12,
    "saving":0,
    "birthDay":20,
    "birthMonth":10,
    "birthYear":2000
  },
  {
    "playerName":"Player Two",
    "position":"defense",
    "captain":false,
    "skating":10,
    "shooting":10,
    "checking":10,
    "saving":0,
    "birthDay":20,
    "birthMonth":10,
    "birthYear":2000
  },
  {
    "playerName":"Player Three",
    "position":"goalie",
    "captain":false,
    "skating":10,
    "shooting":4,
    "checking":9,
    "saving":18,
    "birthDay":20,
    "birthMonth":10,
                    "birthYear":2000
                  }
                ]
              }
            ]
          }
        ]
      }
    ],
    "freeAgents":[
      {
        "playerName":"Agent One",
        "position":"forward",
        "skating":10,
        "shooting":10,
        "checking":10,
        "saving":0,
        "birthDay":20,
        "birthMonth":10,
        "birthYear":2000
      },
      {
        "playerName":"Agent Two",
        "position":"defense",
        "skating":10,
        "shooting":10,
        "checking":10,
        "saving":0,
        "birthDay":20,
        "birthMonth":10,
        "birthYear":2000
      },
      {
        "playerName":"Agent Three",
        "position":"goalie",
        "skating":10,
        "shooting":5,
        "checking":5,
        "saving":0,
        "birthDay":20,
        "birthMonth":10,
        "birthYear":2000
      }
    ],
  "coaches":[
    {
      "name":"Joe Smith",
      "skating":0.5,
      "shooting":0.8,
      "checking":0.3,
      "saving":1.0
    },
    {
      "name":"Frank Smith",
      "skating":0.5,
      "shooting":0.8,
      "checking":0.3,
      "saving":0.5
    },
    {
      "name":"Samantha Smith",
      "skating":1.0,
      "shooting":0.5,
      "checking":0.5,
      "saving":0.0
    }
  ],
  "generalManagers":[
    {
      "name":"Raghav Cherry",
      "personality":"gambler"
    },
    {
      "name":"Joseph Squidly",
      "personality":"normal"
    },
    {
      "name":"Tom Spaghetti",
      "personality":"shrewd"
    }
  ]
}
```

Note the following:

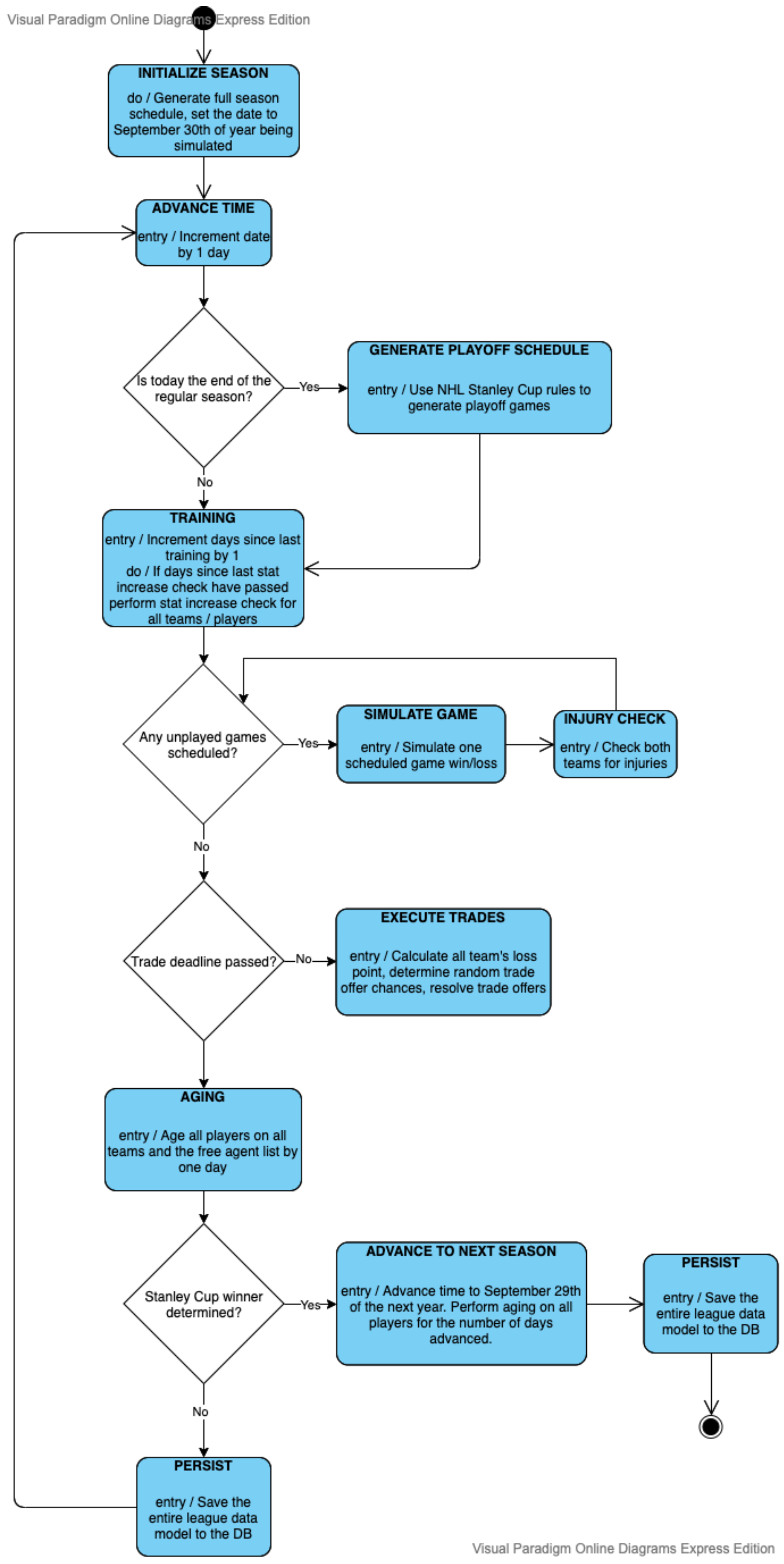
- []'s denote arrays. JSON does not specify array dimensions. You should assume that a league can have any even number of conferences, each conference has an even number of divisions, and each division can have any number of teams. Teams have 20 players.

- Where you see a string value, assume that it should have validation to ensure it is not an empty string.
- "position" items should have values: "goalie", "forward" or "defense" only
- "captain" must be true or false, only one player can be captain on a team
- "freeAgents" is a list of players not assigned to a team that can be hired to fill holes on teams caused by team creation, retirement or trading.

Appendix B – State Machines



Visual Paradigm Online Diagrams Express Edition
File arg given?
No
Yes
IMPORT
entry / validate JSON
do / instantiate & configure league object model
LOAD TEAM
entry / prompt for a team name to load
do / load league data model to memory for given team
CREATE TEAM
do / Prompt for team name, display & choose general manager, display coaches & choose head coach, display free agents & choose team roster
PLAYER CHOICE
entry / Prompt for number of seasons to simulate
do / wait for response
exit / pass number of seasons to simulate state
SIMULATE
entry / Simulate number of seasons passed into state (see nested season state diagram)
Visual Paradigm Online Diagrams Express Edition


Visual Paradigm Online Diagrams Express Edition
INITIALIZE SEASON
do / Generate full season schedule, set the date to September 30th of year being simulated
ADVANCE TIME
entry / Increment date by 1 day
Is today the end of the regular season?
Yes
GENERATE PLAYOFF SCHEDULE
entry / Use NHL Stanley Cup rules to generate playoff games
No
TRAINING
entry / Increment days since last training by 1
do / If days since last stat increase check have passed perform stat increase check for all teams / players
Any unplayed games scheduled?
Yes
SIMULATE GAME
entry / Simulate one scheduled game win/loss
INJURY CHECK
entry / Check both teams for injuries
No
Trade deadline passed?
No
EXECUTE TRADES
entry / Calculate all team's loss point, determine random trade offer chances, resolve trade offers
AGING
entry / Age all players on all teams and the free agent list by one day
Stanley Cup winner determined?
Yes
ADVANCE TO NEXT SEASON
entry / Advance time to September 29th of the next year. Perform aging on all players for the number of days advanced.
PERSIST
entry / Save the entire league data model to the DB
No
PERSIST
entry / Save the entire league data model to the DB
Visual Paradigm Online Diagrams Express Edition

Notes:

The UML state diagrams above represent the state machines for the simulation for milestone 2. Every state lists actions that occur when the state is entered (entry), the action performed in the state (do) and the actions that occur when the state is exited (exit).

The first state diagram shows the state machine for the entire system. This has changed little from milestone 1 other than to factor in the changes we made midway through milestone 1.

The second state diagram shows the internal state machine you must nest inside the **SIMULATE** state.

INITIALIZE SEASON

This state initializes the schedule for the regular season. It sets up whatever classes you have designed to track time and scheduled games. It also sets the current date to September 30th so that when the simulation begins we will begin on October 1st, the first day of the regular season.

ADVANCE TIME

This state advances time by 1 day.

GENERATE PLAYOFF SCHEDULE

This state generates the playoff schedule of games at the end of the regular season. It determines the league rankings and then seeds the playoff spots as per the rules of the Stanley Cup.
https://en.wikipedia.org/wiki/Stanley_Cup_playoffs

TRAINING

This state tracks the time elapsed since the system last performed a training check on all of the players. Remember that free agents are not trained because they are not on a team. Only players assigned to a team when training is calculated can potentially receive stat increases.

SIMULATE GAME

This state simulates the results of a game. Previous milestones used a simplistic formula for determining results, in this milestone the process becomes much more complicated.

INJURY CHECK

Check all players involved in the game for injuries.

EXECUTE TRADES

Evaluate loss points and generate trade offers.

AGING
All players in the league age by 1 day. This will slowly heal injuries and progress players towards retirement.

ADVANCE TO NEXT SEASON
Once the Stanley Cup is resolved we can simulate directly to September 29th, one day before the beginning system date. We need to do this in order to correctly age the players outside of the regular season.

PERSIST
To avoid loss of data we should persist the state of the league after each day of simulation and at the end of simulating a season of play. This state instructs the entire league data model to persist itself through whatever mechanism you create.

Appendix C - Draft Rules

The goal of drafting is twofold: to bring new players into the league and replenish retiring players and to improve poorly performing teams by giving them the chance to pick from new players first.

When?

Perform the draft on July 15th every year in the simulation.

Selection Order (Picks)

We will not follow the NHL draft system. It uses a complicated lottery system that is silly. The selection order, or "draft pick", of the teams is determined by regular season standing, and playoff results.

1. Order teams by their regular season standings, sorted in reverse (worst team to best team)
2. Picks **1 to n** (where n = total number of teams – 16) are the first **n** teams to pick (in other words, the worst teams pick before good teams, and the worst team of all picks first).
3. Order the remaining 16 teams by their Stanley cup playoff standings, sorted in reverse (the first team eliminated at the top, the Stanley cup winner at the bottom)

Rounds

There will be **7** rounds of drafting. This means there needs to be **7 x # Teams** players to draft.

Generating Players

Generate players randomly. Players being drafted are between 18 and 21 years old. Generate their names from a large array of first and last names, randomly.

50% of new players should be forwards.
40% of new players should be defense.
10% of new players should be goalies.

When generating random stats consider the player's position. Generally we don't draft bad players, therefore skew the randomness in the player's favour:

Stat	Forward	Defense	Goalie
Skating	12 – 20	10 – 19	8 - 15
Shooting	12 – 20	9 – 18	1 - 10
Checking	9 – 18	12 – 20	1 - 12
Saving	1 – 7	1 – 12	12 - 20

Appendix D – Game Simulation Notes

You must write an algorithm that accomplishes these goals to the best of your ability. There are multiple approaches to this problem. One approach is to treat the game like another nested level of state machine where you resolve goal scoring attempts on both sides, say 10 seconds at a time or something like that. Another approach if you are good at Math is to resolve all of these constraints into a statistical formula.

Try to think of behavioural patterns that may help you in the implementation of your algorithm. For example, consider the Strategy pattern if you have a generic algorithm but different variant solutions depending on the current state of the game. Do you have something you need each of your Player objects to decide? Consider Template Method. Etc. Do not use patterns inappropriately, but review the catalog before beginning your work to ensure you don't miss potential existing solutions.

Whatever approach you pick is fine with me so long as you achieve these goals:

- Games consist of 3 periods, each 20 minutes long.
- Divide the time on the ice into 40 "shifts", shifts are moments in time where each team has 3 specific forwards, 2 defensemen and a goalie on the ice.
- The average forward or defensemen plays for 18 minutes of the game (in shifts, so they aren't always on the ice, they alternate constantly). Balance the available shifts across the 20 players on the **active** roster.
- Teams with forwards with high shooting stats produce more shots on the opposing goal. The average number of shots on a goal in an NHL game is 30. Increase or decrease the average number of shots on a goal per game up to a maximum of 10 in either direction by the difference between the offensive team's skating stats and the defending team's skating stats. Fast teams shoot more!
- Teams with defensemen with high checking stats reduce the number of shots on their own goal, at the risk of triggering penalties. When a shot is prevented, use the new **penaltyChance** value from your app config to determine the odds of prevention resulting in a penalty.
- When penalties occur the team loses a player for 2 or 5 minutes, we'll just use the 2 minute penalties and ignore the 5 minute ones. This creates 5 vs. 4 power play opportunities for the opposing team, generally resulting in greater odds of scoring. Track this time and factor it into the algorithm's calculations. **Randomly pick one defensemen to receive the penalty.**
- Goalies with high saving stats are more likely to prevent a shot from becoming a goal. We'll say that in regular season games any goalie can only play 2 periods, the backup goalie plays the other period. In playoff games the team's best goalie plays the whole game unless they are injured.
- When a team scores, pick a random forward from the scoring team's current shift to be the goal scorer.

- When a penalty occurs, pick a random defensemen to be the cause of the penalty. This player's stats will be missing from any calculation while they are serving their penalty.
- When a goal is prevented this counts as a save for the current goalie.

Evaluation of your algorithm

These are the average statistics for a game for one team in the NHL (so double it for the game total stats):

- Goals per game: 3.1
- Penalties per game: 2.97
- Shots: 31.3
- Saves: 28.4

To consider your requirement met your algorithm should produce similar results. We will verify this by observing the values output during your simulation. Remember it must do so in a way that the work (scoring goals, triggering penalties, preventing goals / save) is attributable to individual players. This will be important for the programmer working on the trophy epic.

Appendix E – Trophies

President's Trophy

Awarded to the **Team** with the highest points at the end of the regular season.

Calder Memorial Trophy

Awarded to the **best player** in the league where it is there first year in the NHL (they were drafted in the previous summer).

Vezina Trophy

Awarded to the **best goalie** in the league at the end of the Stanley Cup Playoff.

Jack Adam's Award

Awarded to the **best coach** in the league, as determined by the best stat improvement results in training.

Maurice Richard Trophy

Awarded to the **top goal scorer** in the league at the end of the Stanley Cup Playoff.

Rob Hawkey Memorial Cup

I love the violence of hockey! Awarded to the **best defensemen** in the league at the end of the Stanley Cup Playoff. In my opinion the best defensemen is the defensemen that earns the most penalties!

Participation Award

Awarded to the **Team** with the lowest points at the end of the regular season.

Resources

National Hockey League:
https://en.wikipedia.org/wiki/National_Hockey_League

This video shows the NHL'06 Dynasty Mode in action, if you are still uncertain what exactly we're building here this will give you a good idea of what a fully functioning version of what we are building would do and look like. We are not building the interface for this system, but the back end logic that drives it.
https://www.youtube.com/watch?v=lJPppnVbTpU

Stanley Cup Playoff Format
https://en.wikipedia.org/wiki/Stanley_Cup_playoffs