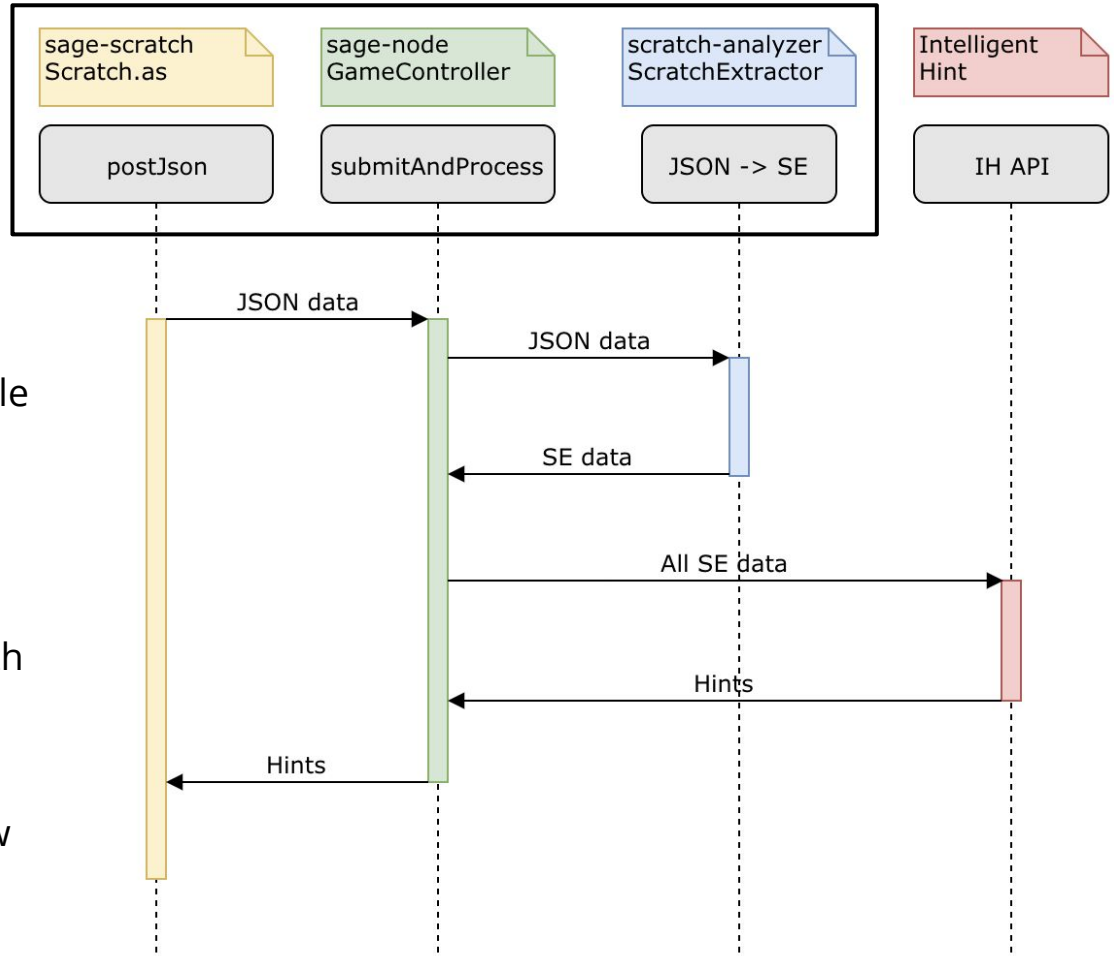
Intelligent Hinting

Robby Costales, Alina Ying, Eddie Shen, Lily Yu Li

Intelligent Hinting Integration
Lily

Data Flow

1. Sage-scratch sends the game snapshot in JSON to sage-node on a mouse click.
2. Sage-node uploads JSON as a file to the database.
3. Sage-node converts JSON to SE using ScratchExtractor.
4. Sage-node sends all SEs to Intelligent Hinting.
5. Intelligent Hinting responds with updated hints.
6. Sage-node forwards hints to sage-scratch.
7. Sage-scratch processes the new hint information.



Node APIs

- /games/uploadJson/:studentID/:gameID/:objectiveID
  - Uploads a JSON string for a game snapshot.
- /games/uploadSe/:studentID/:gameID/:objectiveID/:hasBlockIds
  - Uploads an SE string for a game snapshot.
  - The SE string contains block IDs if :hasBlockIds is true.
- /games/downloadSe/:studentID/:gameID/:objectiveID/:hasBlockIds
  - Downloads all SE data for a specific game.
  - The SE data will contain block IDs if :hasBlockIds is true.
- /games/jsonToSe/:showId
  - Parses a JSON string to an SE string.
  - The SE string will contain block IDs if :showId is true.
- /games/file/:filename
  - Downloads a file by filename.
- /games/file/delete/:filename
  - Deletes a file by filename.

...

Hint Request & Response

Request example (all SE data in the data flow diagram):


```
{
   "seFiles": [
      {
         "content": "<<Object Stage>>",
         "timestamp": 1544503546582
      },
      {
         "content": "<<Object Stage>>\n\t\twhenGreenFlag",
         "timestamp": 1544503573786
      }
   ],
   "info": {
      "studentID": "stu123",
      "gameID": "game123",
      "objectiveID": "obj123",
      "hasBlockIds": false
   }
}
```


Response example 1:


```
{
   "hints": ["wait:elapsed:from:", "randomFrom:to:"],
   "nextAutoHintTime": 1544503583786
}
```


- Hints will be shown to user at nextAutoHintTime automatically or on demand.

Response example 2:


```
{
   "hints": ["wait:elapsed:from:", "randomFrom:to:"]
}
```

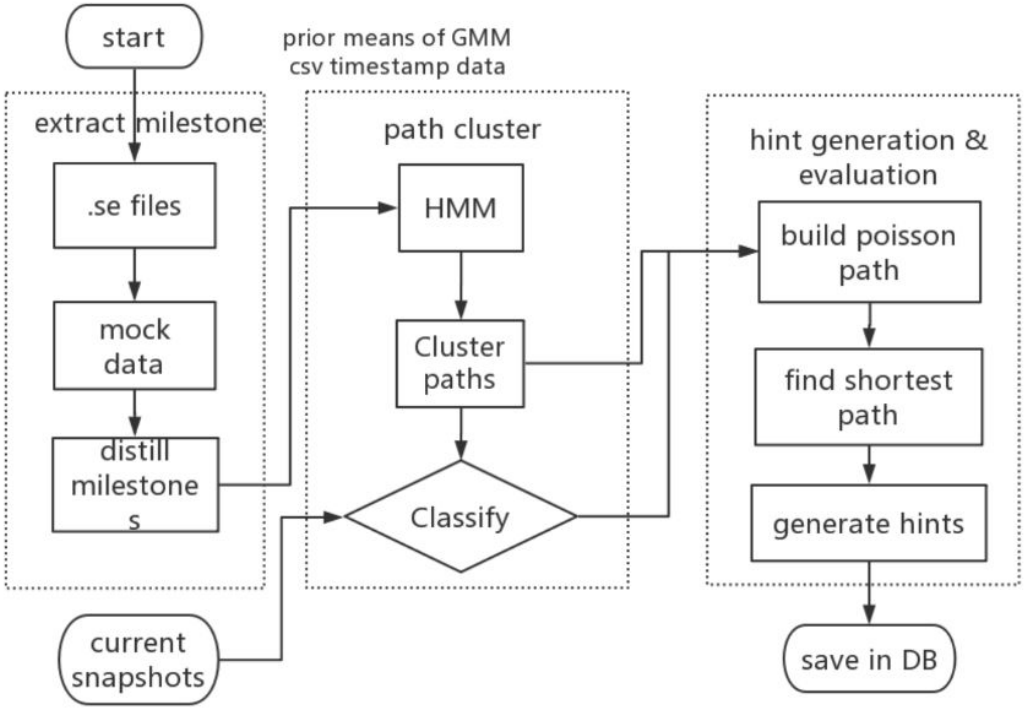

- Hints will be shown to user on demand.

Future Work

- Integration with User Behavior Detection
  - Intelligent Hint also needs a user type
- Save solutions and other supporting files for Intelligent Hint
  - Every time a new game is created, Intelligent Hint needs to create some files that will be used for generating hints for this game.
  - The supporting files and the solution should be included in the hint request as well.
- Support new hint types

Intelligent Hinting
Polishing and Integration
Robby

Previous Work



- Functionality (created in Python) to generate hints for a given snapshot based on mock data generated for that puzzle
- Tested on one of the available puzzles

Limitations of Previous Work

- Lacking documentation
  - Mapping between high level explanation in paper(s) and low level execution in code unclear
  - Issue for current and future contributors
- System offline (not integrated in SAGE) and functions dependent on local file structure, only one game supported (for testing)
- Bugs in code, including ones related to underlying functionality and differing development environments

Semester Priorities + Progress

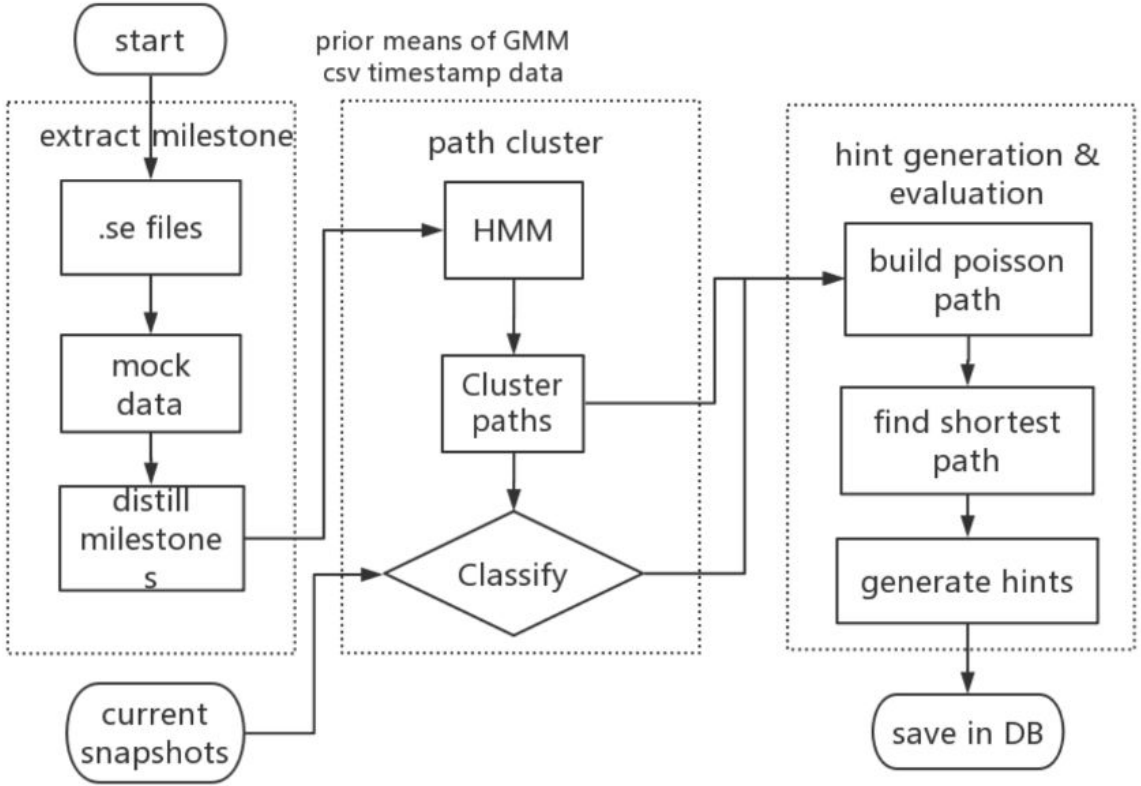
1. Remove local dependencies from code and bring functionality online
   - Finished connection between SAGE node and Intelligent Hint (see demo)
   - Removed dependencies on file structure from algorithms / generalized functionality to any puzzle
     - Local file structure still currently used, but can now be more easily replaced with remote storage (future work)
2. Document and polish current code
   - Bugs fixed
   - Documentation of previous code and new code (in progress--will be finalized alongside completion of written report)

Future Work

- Create system to store algorithm resources (like mock data and/or real data) and outputs in remote database rather than local file structure (also in progress)
- Take historical snapshots into consideration, in addition to current snapshot
- Assess effectiveness of algorithm (can be done with and/or without real student data)

SAGE Mock Data Generation
Alina

Mock Data Generation



Mock data is the quickest way to generate **large amounts** of randomized **data** for data analysis and code algorithm verification (including hint generation)

Mock Data Generation: Summary

Game	8 Ball	2014 Christmas Countdown	Aquarium (Piano Cover)	Cubified Logos	Face Morphing	Fizz- A Platformer	Lune of Hippocrates and Lunes of Alhazen
# of lines	56	617	218	55	681	113	282
Game	Make your own star!	Morphing from a Square to Circle	Morphing Greek alphabet Letters using Bezier curves	Morphing Numbers using Bezier curves	Mouse Maze	Music Waves	Neon Art Contest
# of lines	868	349	611	672	135	78	24
Game	Never Forget	Paint v1.0	Palette tool - Color picker!	Rotating Shapes Interactive	Santa's Sleigh	Scratch Blaster 2.21	Scratch Minigames | Steve Jobs Timeline
# of lines	172	152	69	149	151	4543	563 | 638

Mock Data Generation: Approach

Neon Art Contest (24 loc)

```
<<Object Stage>>
        whenGreenFlag
        doForever
                doPlaySoundAndWait
        <<Object drawing1>>
                whenKeyPressed
                doRepeat
                        changeGraphicEffect:by:
                nextCostume
                doRepeat
                        changeGraphicEffect:by:
                whenGreenFlag
                gotoX:y:
                show
                lookLike:
                doForever
                        doIfElse
                        =
                        costumeIndex
                        gotoX:y:
                        gotoX:y:
        <<Object drawing2>>
                whenGreenFlag
                hide
```

- Games
  - 22 games
- Student types
  - categorized as 4 types of students: stopper, tinkerer, mover, extreme-mover
- Generation of mock data
  - Failure data: fail at a random location, for all games
  - Success data: finish program, but stuck at multiple locations (time series-based)

Mock Data Generation: generate Failures (incomplete finish)

Base Data (with ideal, complete program sequence)

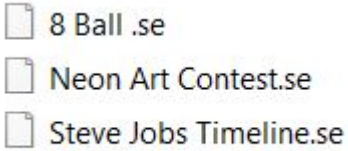
- 8 Ball .se
- Neon Art Contest.se
- Steve Jobs Timeline.se

Mock Data

30 incomplete SE files	
18 incomplete SE files	
94 incomplete SE files	
116 incomplete SE files	

Mock Data Generation: generate Failures (incomplete finish)

Base Data (with complete program sequence)



Mock Data

693 incomplete SE files
454 incomplete SE files
2862 incomplete SE files
4463 incomplete SE files

Mock Data Generation: different Stats for different players

```
def stopper(url, l):
    if not os.path.isdir(url):
        os.makedirs(url)
    i = 1
    count = 0
    while i < len(l):
        interval = abs(int(random.normal(5,3)))
        t = 0
        while t < interval:
            ouf = open(url+str(count)+'.se', 'w')
            ouf.writelines(l[:i])
            ouf.close()
            t += 1
            count += 1
        i += 1
```

```
def mover(url, l):
    if not os.path.isdir(url):
        os.makedirs(url)
    i = 1
    count = 0
    while i < len(l):
        interval = abs(int(random.normal(1,1)))
        t = 0
        while t < interval:
            ouf = open(url+str(count)+'.se', 'w')
            ouf.writelines(l[:i])
            ouf.close()
            t += 1
            count += 1
        i += 1
```

Mock Data Generation: generate Successes (with “stuck” points)

Base Data (with complete program sequence)

- 8 Ball .se
- Neon Art Contest.se
- Steve Jobs Timeline.se

Mock Data

1 SE file, 1 “stuck” moment
1 SE file, 2 “stuck” moments
1 SE file, 4 “stuck” moments
1 SE file, 3 “stuck” moments

Mock Data Generation: generate Successes (with “stuck” points)

```
<<Object Stage>>
        whenGreenFlag
stuck #0        doForever
                doPlaySoundAndWait
        <<Object drawing1>>
                whenKeyPressed
stuck #3                doRepeat
                        changeGraphicEffect:by:
                nextCostume
                doRepeat
                        changeGraphicEffect:by:
                whenGreenFlag
                gotoX:y:
                show
                lookLike:
                doForever
stuck #2                        doIfElse
                        =
                        costumeIndex
                        gotoX:y:
                        gotoX:y:
        <<Object drawing2>>
                whenGreenFlag
                hide
```

```
def tinkerer (url, l):
    if not os.path.isdir(url):
        os.makedirs(url)
    w=l.copy()
    count=0
    while (count<3):
        index1 = randint(0, len(l)-1)
#       index2 = randint(0, len(temp)-1)
        if ("Object" not in w[index1]):
                w[index1]="stuck #"+str(count)+" "+w[index1]
        count+=1
    ouf = open(url+str(count)+'_new.se', 'w')
    ouf.writelines(w)
    ouf.close()
```

Future Work: Translate the “stuck” points into time delay given by the timestamps/time series

SAGE Mock Data Generation: A Failed Attempt

Eddie

Understanding mockdataBD

New mock data format (mover/stopper/extremeMover/tinkerer) :

```
{
   ‘Timestamp’: hr/min/sec/mo/d/yr
   ‘Content’: a, b, c
}
```

- Note that ‘content’ is cumulative !
- We want differential data to see each step

Understanding mockdataBD

mockPerProject (n, se)

@se: an array of games sourced from completeSE

```
[ [game 1], [game 2], … , [game n]]
Where game = se.readlines()
```

@n: number of mocks produced for a given game

Understanding mockdataBD

make_mock_data (num)

@return: project_collections

```
[[game a1], [game a2], … , [game an]
 [game b1], [game b2], … , [game bn]
                …
 [game m1], [game m2], … , [game mn]]
```

Each row is a type of student; each game entails all snapshots

Understanding mockdataBD

Ideally, we want something like this to be stored as .csv files locally:

Timestamp	Operation1	Operation2	…	Operation 138
T1	1	0	…	0
T2	0	0	…	1
…	…	…	…	…
Tn	0	1	…	0

To achieve this we have to One-Hot encode project_collections

Issues with Conversion

```python
#Reads SE file, converts it to CSV file, creates CSV file with all the operations listed
#Generates mock success and failure data
#Create 4 x 25 mock data
if __name__ == "__main__":
    # read_se("sage-frontend/machine_learning/sample_data/0/CTG-22.se")
    # convert_se_to_csv("")
    # calculate("completeSE")
    # for i in range(5):
    #     generate_mock_se_success("Face Morphing.se", i + 1)
    #     generate_mock_se_failure("Face Morphing.se", i + 1)

    operation = read_operation("operations.csv")
    # input_file = "../mockData/failure/1/1/100.se"
    # print(read_se(input_file, operation))

    for i in range(1, 5):
        for j in range(25):
            convert_se_to_csv("../mockSE/success/" + str(i) + "/" + str(j) + "/", i, j)
```

```python
#Converts the SE file to CSV file, with cumulative datapoints
def convert_se_to_csv(file_path, student_type, student_id):
    diff = []
    n = len(glob.glob(file_path + '/*.se'))
    # print(file_path)
    # print(n)
    one_hot_data = np.zeros((n, len(operation)))
    for i in range(n):
        list2 = read_se(os.path.join(file_path, str(i) + ".se"))
        for opt in list2:
            one_hot_data[i][opt] += 1

    one_hot_data = one_hot_data.astype(int)
    df = pd.DataFrame(one_hot_data)

    try:
        os.mkdir("../csv_file/original/success/" + str(student_type))
    except OSError as e:
        # print(e)
        pass

    df.to_csv("../csv_file/original/success/" + str(student_type) + "/output" + str(student_id) + ".csv")
```

- Main problem is missing path, causing confusion of data format
- Theory: n used to represent the number of lines, not number of files

Future Work

- Work based on previously mentioned theory; ignore past work for now and revamp file paths, etc.
- Use the same method of timestamp generation on game block attributes instead of games