Panalyzer

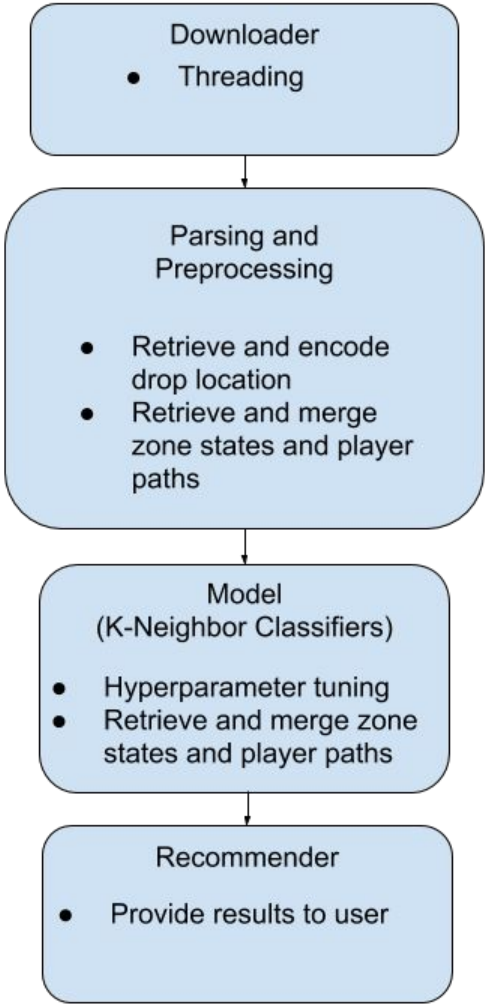
Pat Farrell, James Kao, Samuel Sachnoff,
Andrew LeDawson

Project Definition and Overview

- PUBG is a complex game with many variables, most players will lose each game
- PUBG gameplay review
  - Play on different islands with teams of 1-4, total of 100 players
  - Parachute from plane path onto island
  - Get items like guns and kill each other
  - Shrinking play area forces players together
- Provide recommendations:
  - Drop Location
  - Path to take
  - Items to pick up
  - etc

Implementation

- Language: Python
  - Ease of data manipulation
  - Many relevant libraries
- Libraries:
  - scitkit-learn for machine learning
  - Pandas
  - Matplot
  - Numpy
- APIs
  - PUBG API kit
    - Random matches
    - Match details
    - Match “telemetry”
      - Detailed event log



Data Acquisition

PUBG JSON API

- Telemetry Events and Objects

CHARACTER

```
{
  "name":         string,
  "teamId":       int,
  "health":       number,
  "location":     Location,
  "ranking":      int,
  "accountId":    string
  "isInBlueZone": bool,
  "isInRedZone":  bool,
  "zone":         [regionId, ...]
}
```

COMMON

```
{
  "isGame":  number
},
```

isGame represents the phase of the game defined by the status of bluezone and safezone:

```
isGame = 0 -> Before lift off
isGame = 0.1 -> On airplane
isGame = 0.5 -> When there's no 'zone' on map(before game starts)
isGame = 1.0 -> First safezone and bluezone appear
isGame = 1.5 -> First bluezone shrinks
isGame = 2.0 -> Second bluezone appears
isGame = 2.5 -> Second bluezone shrinks
...
```

```
"_D": string,      // Event timestamp
"_T": string,      // Event type
"common": {Common}
```

LOGARMORDESTROY

```
"attackId":           int,
"attacker":           {Character},
"victim":             {Character},
"damageTypeCategory": string,
"damageReason":       string,
"damageCauserName":   string,
"item":               {Item},
"distance":           number
```

LOGCAREPACKAGELAND

```
"itemPackage": {ItemPackage}
```

LOGCAREPACKAGESPAWN

```
"itemPackage": {ItemPackage}
```

LOGGAMESTATEPERIODIC

```
"gameState": {GameState}
```

Downloader

- PUBG Sample API gives 750 matches to download each time*
  - *lol not actually
- Always sanity check your data source
  - Check for duplicate data
- Thread all the things
  - Majority of delays are filesystem-based
- Consider what would influence your data, and make alternate plans
  - Missing game version field? Separate by date instead!

Ontology


owl:Thing
Map
flightPath
Location
Region
Match
Entity
Match_Stage
State
Zone
teamSuccess


Rules:
Rules
Player(?p), hasLocation(?p, ?l), inRegion(?l, ?r), Region(?r) -> hasPlayer(?r, ?p)
Player(?p), gearLevel(?p, "strong"), Region(?r), dangerLevel(?r, "mediumDanger"), hasPlayer(?r, ?p) -> deathRisk(?r, "mediumRisk")
Player(?p), gearLevel(?p, "strong"), Region(?r), dangerLevel(?r, "lowDanger"), hasPlayer(?r, ?p) -> deathRisk(?r, "lowRisk")
Player(?p), gearLevel(?p, "weak"), Region(?r), dangerLevel(?r, "lowDanger"), hasPlayer(?r, ?p) -> deathRisk(?r, "mediumRisk")
Player(?p), gearLevel(?p, "weak"), Region(?r), dangerLevel(?r, "mediumDanger"), hasPlayer(?r, ?p) -> deathRisk(?r, "mediumRisk")
Player(?p), gearLevel(?p, "weak"), Region(?r), dangerLevel(?r, "highDanger"), hasPlayer(?r, ?p) -> deathRisk(?r, "highRisk")
Region(?r), xsd:int[> "10"^^xsd:int](?players), hasPlayers(?r, ?players) -> dangerLevel(?r, "highDanger")
Region(?r), xsd:int[> "5"^^xsd:int , <= "10"^^xsd:int](?players), hasPlayers(?r,

Ontology(Cont.)

Parsing and Processing

- Each PUBG match's "telemetry" is 15MB of JSON
  - If you thought Chrome ate a lot of RAM with a 500kb webpage...
- Search function for extracting specific data from JSON
  - Keep it simple: iterate through with a for loop, multithread to make it faster
- Preprocess data via encoder
- Conversion to DataFrame
- Drop the extraneous data

Models

- Predict best drop location per map
  - Model: KNeighborClassifier
  - Input: x, y, flight path
  - Output: Predicted rank of the play if dropped at position(x,y)
  - Accuracy: Generally around 50%

- Predict expected rank given game and zone state
  - Model: KNeighborClassifier
  - Input: x, y, safeZone x, safeZone y, safeZone radius, gameState
  - Output: Predicted rank of the player at the position (x, y)
  - Accuracy: ~30-40% for most maps - still better than random

Path Recommendation

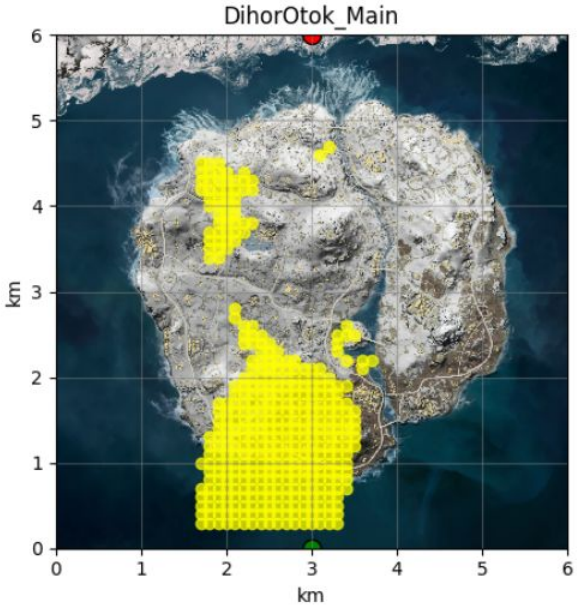
- Uses recommended drop location as starting point
- Look at all neighboring positions, select one with the best predicted rank
  - In case there are multiple with same rank, select a random one

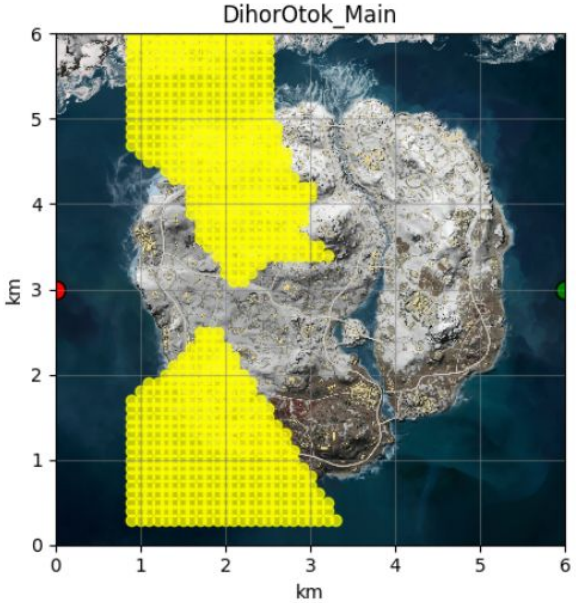
- Iterate until game state is at the end

Testing

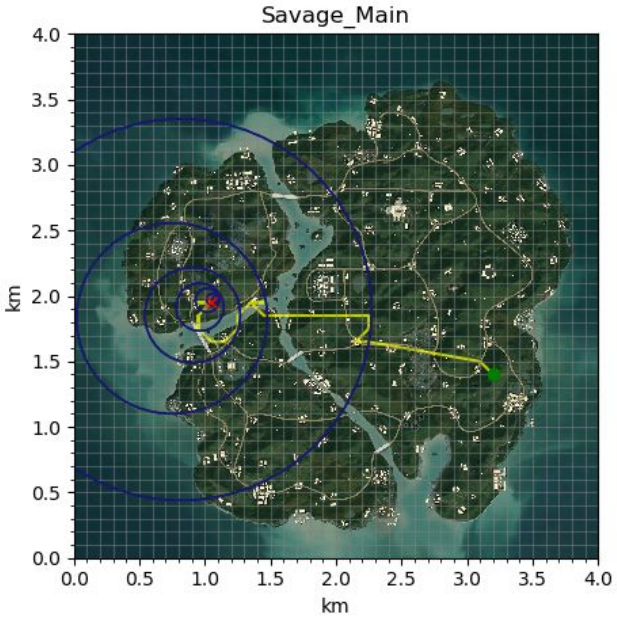
- Cross Validation on all models
- Limited testing with real players
  - We focused on improving our processing and models
  - No interface to generate advice on-the-fly
- Difficult to test with large population
  - Small sample

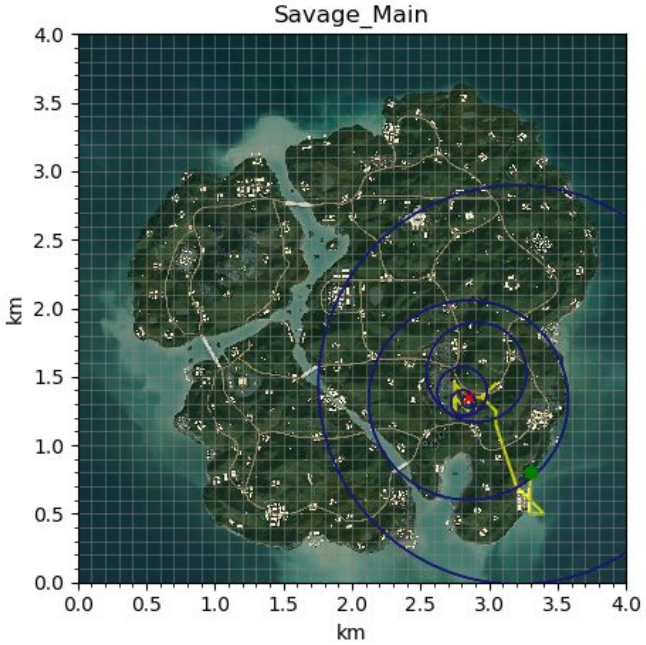
Drop Recommendation Results



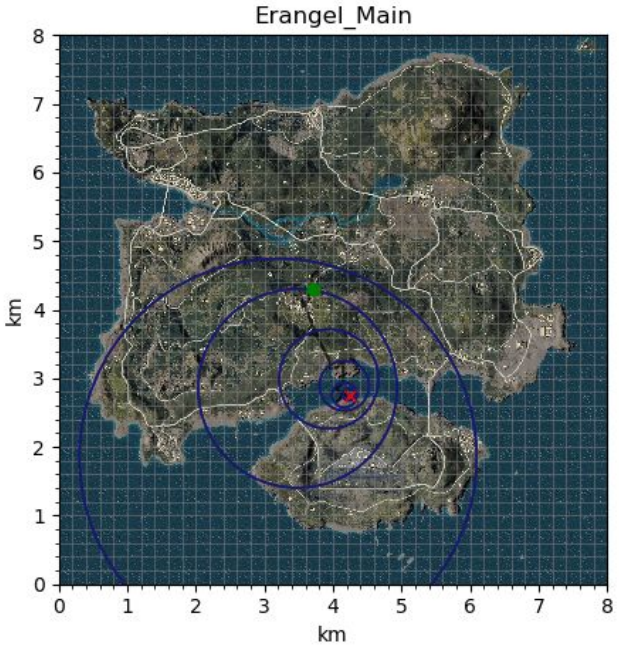


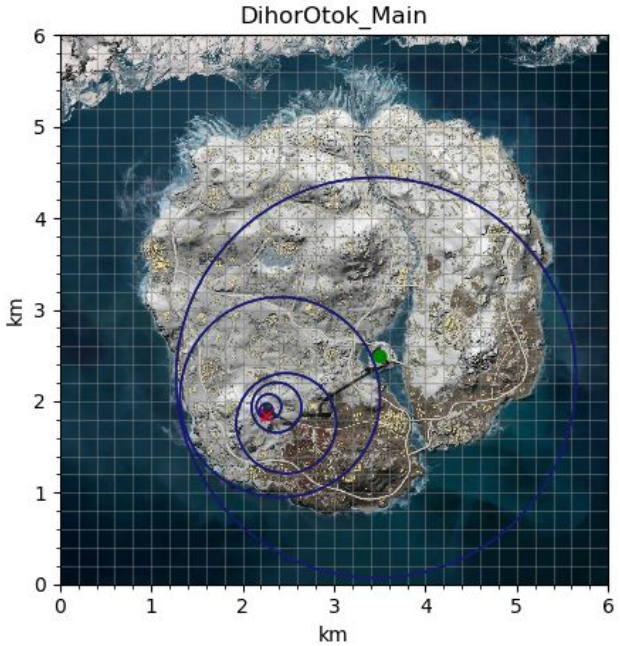
Path Recommendation Results: The Good



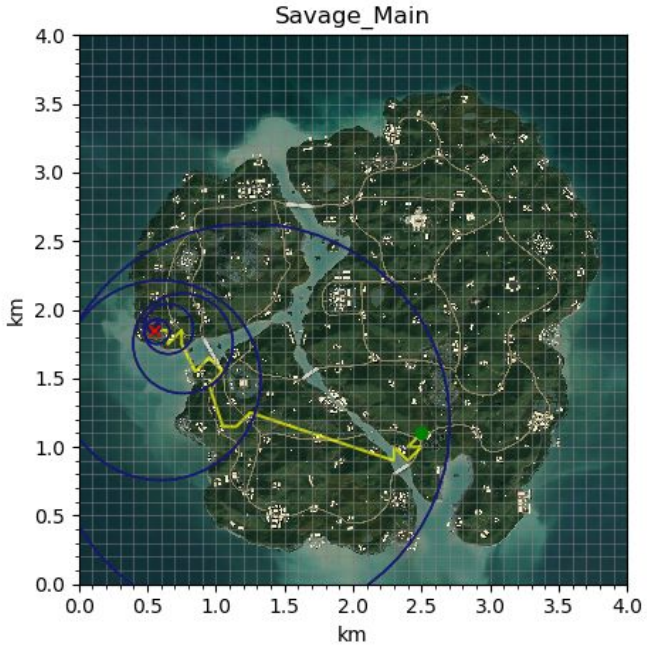


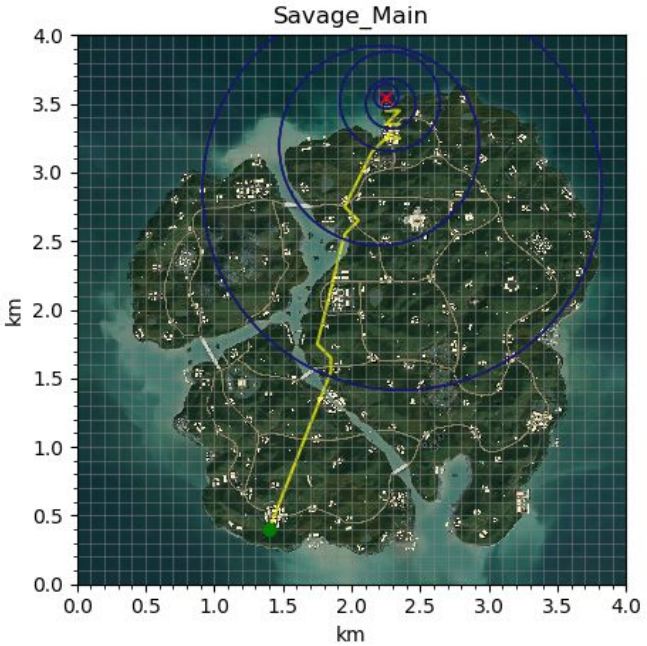
Path Recommendation Results: The Good



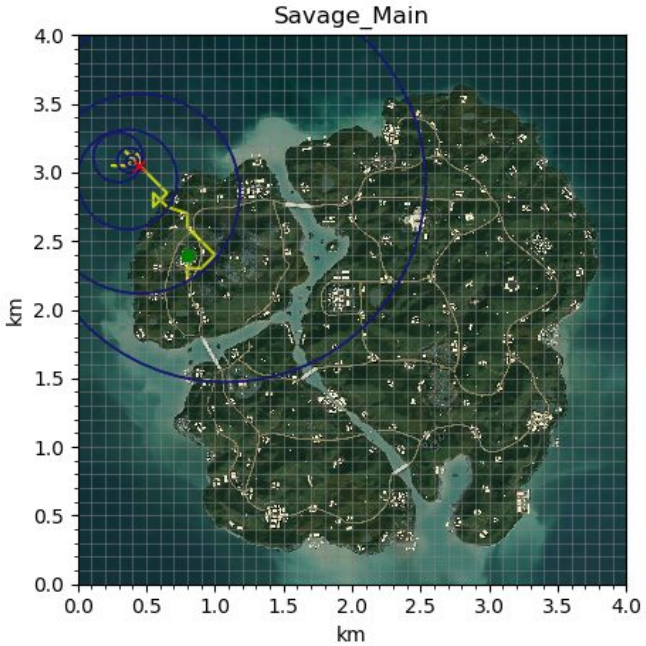


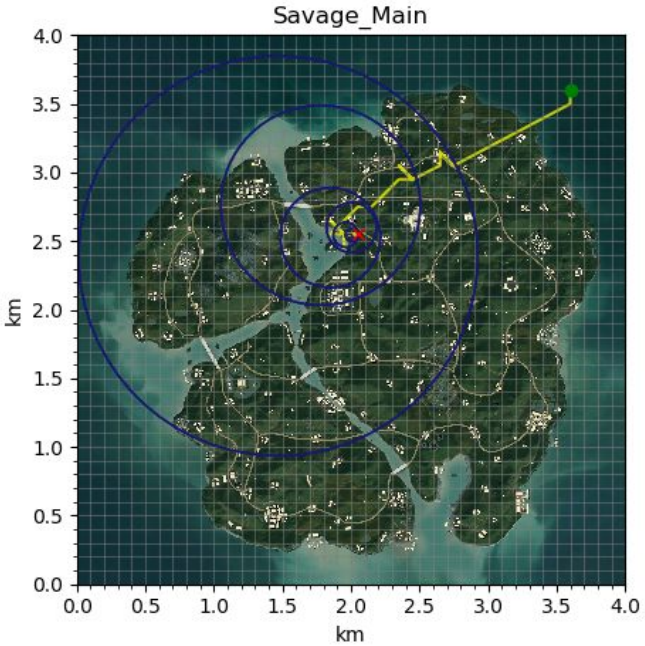
Path Recommendation Results: The Bad





Path Recommendation Results: The Ugly





Next Steps

- Integrate both systems together
- Continue onto other predictions
  - Item Pick Up
  - Vehicle Usage
  - etc
- UI/UX
- Automatically gather matches and update models
- Update code/ontology with more nuance
  - Center of zone cannot be over water

Questions?