## *A. FightingICE Platform*

In this section, FightingICE's game rules, structure and game flow is introduced. FightingICE is a Java based platform with a two-player 2D fighting game that enables developers to implement their own AI controllers, which are able to run any AI algorithm on non-playing characters (NPCs) and make unique attacks and combos. It accepts Java-based and Python-based agents, because Py4J wraps it for Python. The 2017 version of FightingICE also supports development of visual-based AI controllers. It is developed for research purposes by Intelligent Computer Entertainment Lab. at Ritsumeikan University in the manner of their Fighting Game AI Competition. Platform uses The Rumble Fish 2's game sources and currently, three NPCs with different skill sets are available: ZEN, GARNET and LUD. It can be played through a control pad, a keyboard or an AI agent. Developers of the platform emphasize the equal footing of human players and AI bots [7]. Therefore in each frame, agents get information about the game state, process the information and output the keys pressed, same as a human player. Game engine also creates an illusion of slow perception by sending 15 frames delayed game information to the AIs. However, to promote more use of visual-based AIs using deep learning et cetera, this delay constraint can be gotten around via developing a visual-based AI. One game consists of three 60-seconds-rounds. A frame is 1/60 seconds (approx. 16.67 ms). Players should complete the mentioned outputting circle in a frame-time. If no input is generated in a frame, AILoader considers the last input as the new one. This creates the real-time decision making challenge.

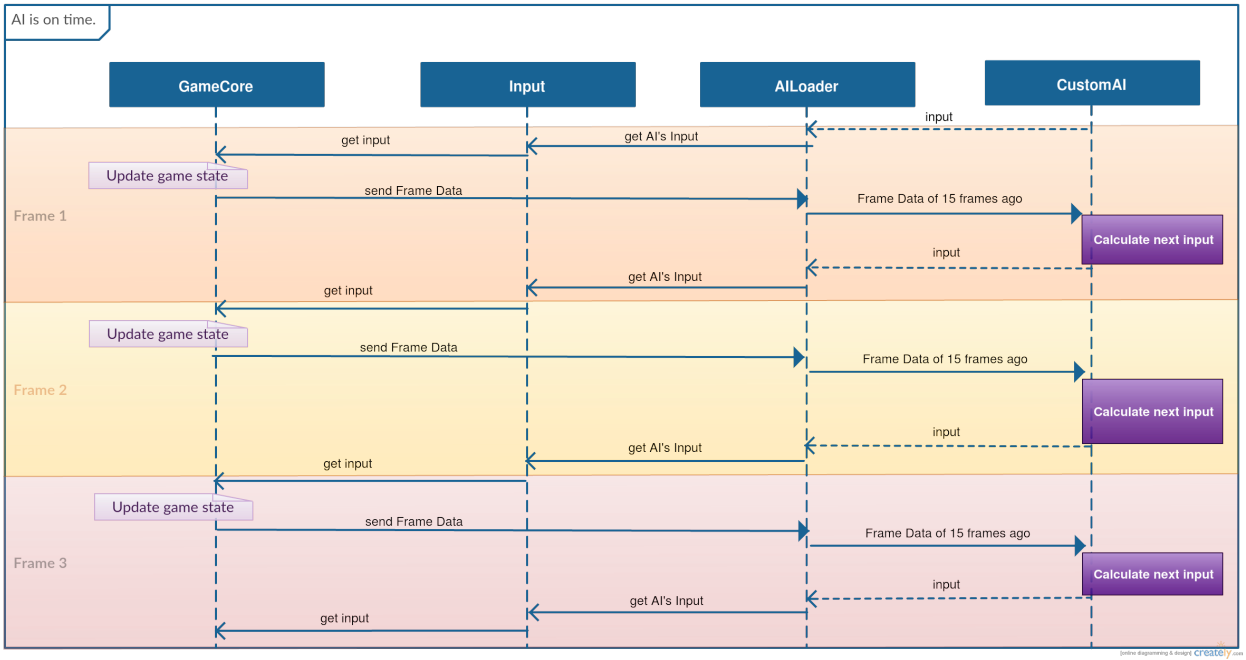



**Figure 1 - Agent responds to AILoader in dedicated time**

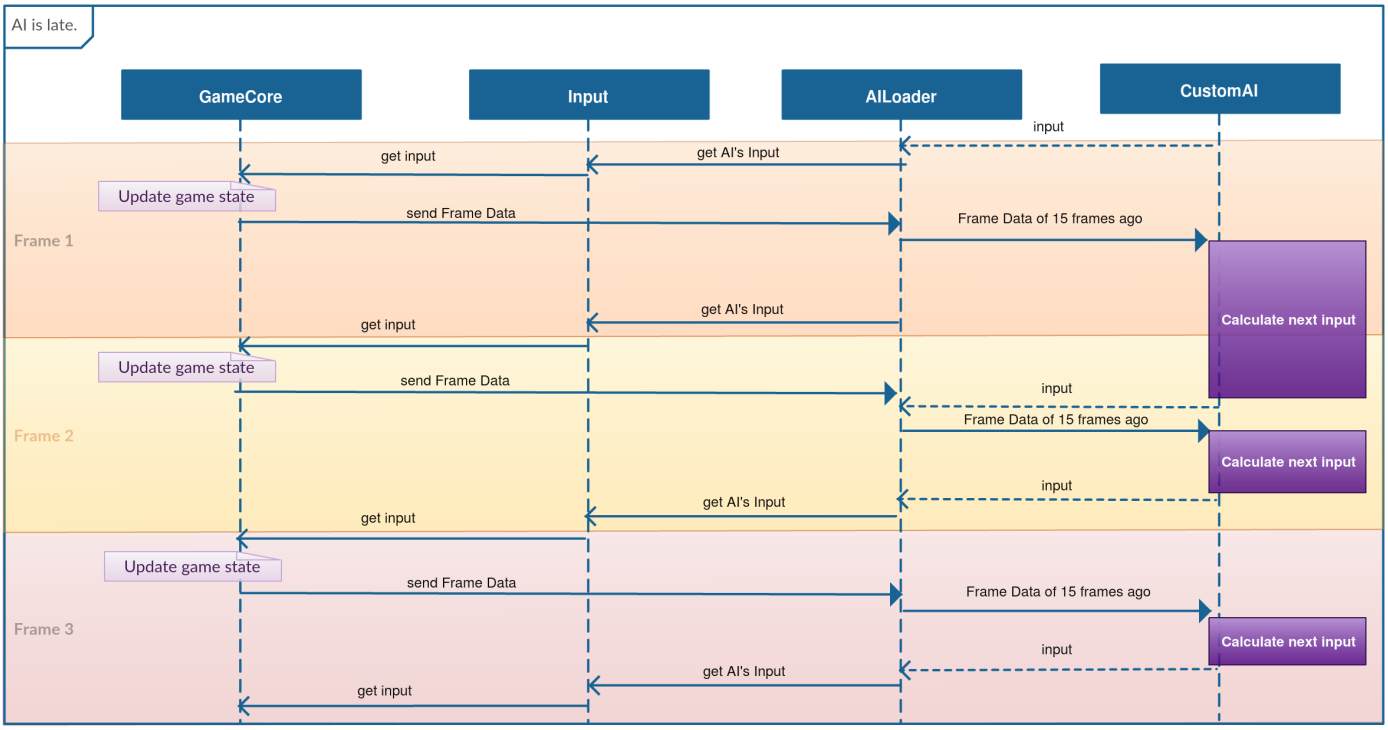



**Figure 2 - Agent cannot respond to AILoader within 16.67 ms**

There are more restrictions: multi-threading is not allowed, for each round players have 5 ms initialization time, the run-time memory can be maximum 512 MB, agents can read and write in /data/aiData/{agentName} but any file written cannot exceed 10 MB.

At the end of each round the score is calculated as follows:

$$Score = \frac{Opponent's\ Hit\ Point}{Player's\ Hit\ Point\ + Opponent's\ Hit\ Point} \times 1000$$

Character based information that can be gathered from the engine is as follows:

- **hp:** Hit point. Starts from zero and decreases when the opponent hits the player.
- **energy:** Starts from zero; increases when the the player hits the opponent, decreases when the opponent hits the player. Energy required skills consume energy after each use.
- **front:** Character's facing directions; true for right, false for left.
- **state :** Four types of states are available: stand, crouch, on air, down
- **action:** Five types of actions are available; base, move, guard, recovery, and skill. Base actions are neutral actions such as standing and crouching. Move actions causes position changes. Guard actions help reducing damage. Recovery actions arise automatically after a hit or a jump. Skill actions are attacks and can generate attack objects like fist, fireball.

- **control:** Controllable flag. While a character is in uncontrollable state, all inputs are neglected. During recovery actions and noncancellable parts of skill actions, characters become uncontrollable.
- **attack:** Generated attack objects by skills.
- **remainingFrame:** Current actions remaining frames.
- **x, y:** Coordinates of character box's top-left corner, where the origin is the top left of the stage.
- **left, right, top, bottom:** Hit box's boundaries.
- **speedX, speedY:** Character's moving speeds.

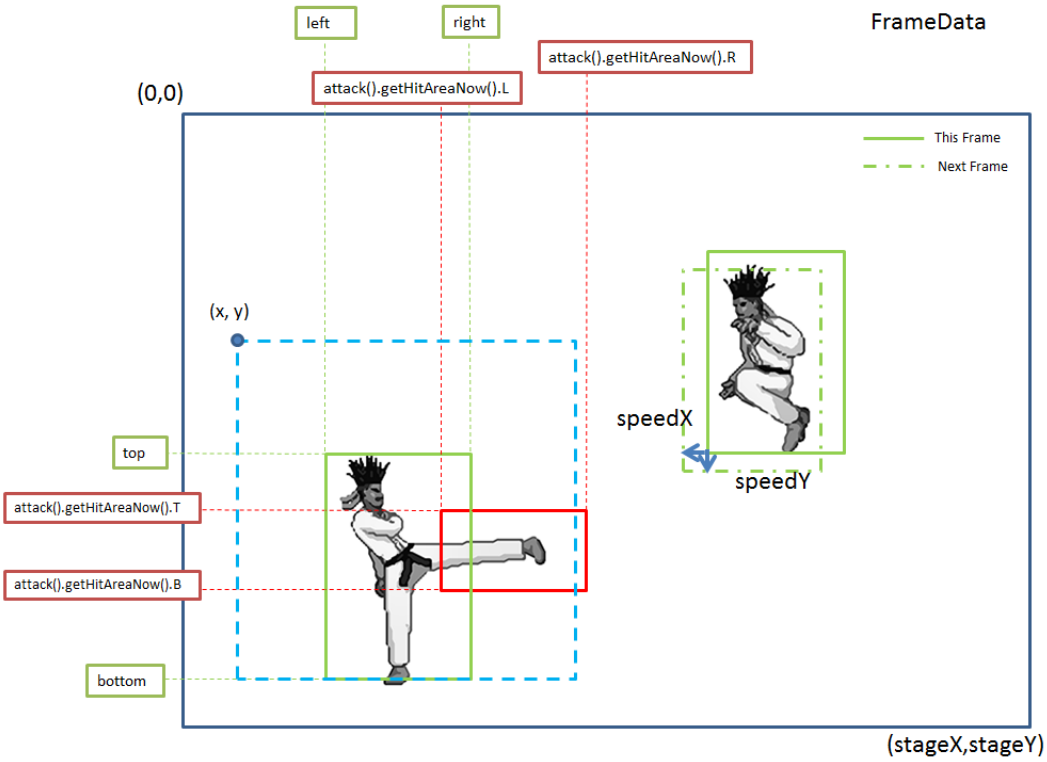



**Figure 3 – Details of character position data [4]**

Each skill is composed of startup, active and recovery stages. As shown below, some skills have a bit of their recovery parts cancellable. As said before, it can be checked via controllable flag within character's data.

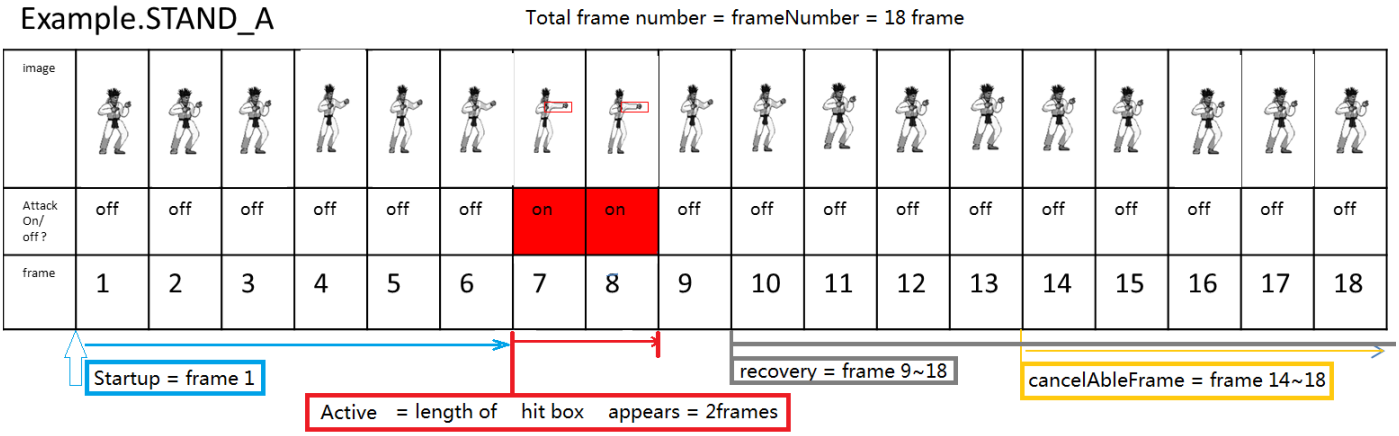



**Figure 4 – Different states of character when a skill is used [4]**

Hit boxes appear in Active frames and with the collusion of an attack hit box and opponent's hit box; player's energy increases, opponent's hit point decreases. Hit boxes disappear in recovery frames. In cancellable frames, any skill that has a lower MotionLevel than the CancellableMotionLevel of the current one can cancel the current skill [4].

Damages, required energies and gains may differ from character to character even for the same action. Below is an example of this differentiation.

| Character | Skill | Command | Damage | Attack Type | Special | Startup | Active | Recovery | Energy Req. | Energy Earned |
|---|---|---|---|---|---|---|---|---|---|---|
| ZEN | STAND_D_DF_FC | 236_C | 300 | high | projectile | 15F | - | 33F | 300 | 30 |
| GARNET | STAND_D_DF_FC | 236_C | 300 | low | - | 6F | 40F | 33F | 300 | 30 |

**Figure 5 – An example of different properties on same action**

Attacks can be blocked by guard actions according to their attack types. The conversion table used for deciding the success of a block can be seen below.

| Guard action / Attack type | STAND_GUARD | CROUCH_GUARD | AIR_GUARD |
|---|---|---|---|
| High | block | block | block |
| Middle | block | hit | block |
| Low | hit | block | hit |
| Throw | hit | hit | miss |

**Figure 6 – Guard to Attack Conversion Table [8]**

2017 Fighting Game AI Competition has six leagues that will run in this order: Zen Standard, Zen Speedrunning, Garnet Standard, Garnet Speedrunning, Lud Standard, and Lud Speedrunning. This is basically, a Standard league and a Speedrunning league for each of three characters ZEN, GARNET and LUD.

Winner of a round in the Standard league considered as the one with a positive HP when its opponent's has reached zero. Both AIs will be given the initial HP of 400. The league for a given character type is conducted in a round-robin fashion with two games for any pair of entry AIs switching P1 and P2. The AI with highest number of winning rounds becomes the league winner.

In the Speedrunning League, the league winner of a given character type is the AI with the shortest average time to beat their sample MctsAi. For each entry AI, 5 games are conducted with the entry AI being P1 and MctsAi being P2, then 5 games more with P1 and P2 is vice versa. The initial HP of MctsAi is set to 300 while that of each entry AI to 9999. If MctsAi cannot be beaten in 60s, the beating time of its opponent entry AI is penalized to 70s.

Using the Formula-1 scoring system, contestants get their points according to their position for each tournament. The competition winner is the one who has the greatest sum of points across all six tournaments [6].

There are multiple arguments available when running the game environment. Below ones are used frequently for stated reasons.

| | |
|---|---|
| Round robin all agents without game display | --black-bg --inverted-player 1 --all --disable-window |
| Open port 4242 and wait for python to start a game | --py4j --port 4242 --black-bg --inverted-player 1 --disable-window |
| Run 5 Zen to Zen games between Machete and Thunder01 | -n 5 --c1 ZEN --c2 ZEN --a1 Machete --a2 Thunder01 |

## *B. Machine Learning Topics*

Machine learning is the construction of algorithms that can make predictions on data after observing it. The idea is to overcome static programming of the machine and using data driven predictions instead [10].

Gaussian Process for Regression is one of the algorithms in machine learning and it belongs to supervised learning. Supervised learning is basically learning from the labeled training data whereas the unsupervised learning uses unlabeled data. Regression, on the other hand, is a supervised learning that aims fitting data into a model and make predictions upon it. Gaussian Process is defining a Gaussian distribution over functions. Just as a Gaussian distribution, it is fully specified by its mean and covariance matrix. A Gaussian process is specified by a mean and a covariance function [9]. Covariance is a function C(x, x') that expresses the expected covariance between the values of the function y at the points x and x'. Covariance function is also called kernel.

Radial Basis Function Kernel is also called Gaussian kernel. In particular, it is commonly used in support vector machine classification [15]. The RBF kernel on two samples x and x', represented as feature vectors in some input space, is defined as below.

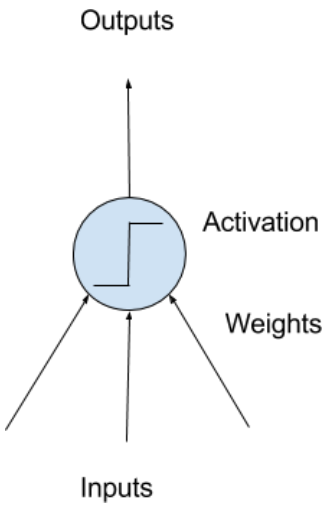
$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$$

Since with the increase of distance the value for RBF kernel increases, RBF can be interpreted as a similarity measure.

Maximum normalization is rescaling attributes to the range of zero to one by that making the largest value for each attribute one and the smallest value zero [14].

K-Fold Cross Validation is splitting the data up into a set of k folds; taking each k-1 fold as train data and the other one as test data, then repeating this k times. This way it will be ended up with k error estimates that can be averaged to a better estimate of error [16].

Artificial neural networks are often just called neural networks. It is inspired by biological neural networks. The ability to learn the representation of the data and how to best relate it to the output variable gives its power to neural nets. Mathematically, neural networks are capable of learning any mapping function and have been proven to be a universal approximation algorithm.




On the side there is a neuron. It is a simple computational unit that has weighted input signals and produces an output signal using an activation function.

The weighted inputs are summed and passed through an activation function which is called an activation function because it governs the threshold at which the neuron is activated and strength of the output signal. With nonlinear activation functions such as sigmoid, hyperbolic tangent and rectified linear unit, a network can combine inputs in very complex ways and therefore its modeling capability increases.

Loss function or cost function returns a number representing how well the neural network performed in mapping training examples to correct output.

Optimizer is the mathematical model that tries to minimize loss function by updating weights. Back propagation is one of them. Gradient Descent is also one of the back propagation algorithm. It is about taking the derivative of loss function and finding the minima.

Sigmoid function bounds the output between 0 and 1 therefore in classification jobs it is used at the most end usually.

Rectified Linear Unit is shown to lead to better results than other activation functions. The reason behind is that when it's gradient is taken it goes to zero that stops learning. On the other hand, rectified linear unit's gradient doesn't go to zero when x is greater than zero.

Regularization is one of the methods used for overcoming overfit problem. For e.g. L2 regularization adds an L2 penalty equal to the square of the magnitude of weights. It is generally added to the loss function.

Precision and Recall is an evaluation metric that is calculated with the help of confusion matrix. Confusion matrix has predictions in one side and ground truth in other side. The matrix elements are the number of occurrences that A, truth, is predicted as B. It is easier to see True Positives (TP), False Positives (FP) and False Negatives (FN) in this fashion.

$$\pi_i(Precision) = \frac{TP_i}{TP_i + FP_i}, \qquad \rho_i(Recall) = \frac{TP_i}{TP_i + FN_i}$$

F-measure is the harmonic mean of Precision and Recall. In most situations, you have a trade-off between precision and recall. If you optimize your classifier to increase one and disfavor the other, the harmonic mean quickly decreases. With unbalanced classes that one class is more important that the other, F-measure of the important classes tells more about the performance of the model.

The softmax classifier is a linear classifier that uses the cross-entropy loss function.

Principal Component Analysis is a dimensionality reduction method. In this method, variables are transformed into a new set of variables, which are linear combination of original ones. These new set of variables are known as principle components. They are obtained in such a way that first principle component accounts for most of the possible variation of original data after which each succeeding component has the highest possible variance.

### *C. Tools and Libraries*

Weka Data Mining Tool is a data-mining tool that included a wide range of machine learning algorithms. The used version for this work is Weka 3.8.0 [3]. It has a very well designed user interface, which can be used to run machine-learning algorithms on the CSV or ARFF formatted data. Numerous algorithms are available for e.g. Logistic Regression, Random Forest, Naïve Bayes, and Multilayer Perceptron, Gaussian Process. In addition to that, Weka can be called from Java code; it has an API.

Miniconda is a package manager for python. It is easier to manage different versions of python with each has different packages by the help of environments in miniconda.

Tensorflow is a software library that is used for numerical computation using data flow graphs. It is very popular in deep learning field. Its flexible architecture allows you to deploy computation to one or more CPUs or GPUs with a single API.

Pandas is a python library that is mainly used for loading, manipulating and analyzing data. It offers data structures and operations for manipulating numerical tables and time series.

Numpy is a highly used library that supports for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays.

Scikit-learn is a Python module for machine learning built on top of SciPy that is a collection of mathematical algorithms and convenience functions built on the Numpy extension of Python.

# Problem Statement

The aim of the project is to design and implement an artificial intelligent that is able to play a fighting game wisely and eventually has a better performance than the rest of the Fighting Game AI Competitors. The main challenge is to design an algorithm about taking action decisions in a real-time environment. In this manner, a deep understanding of the game environment and its progress through possible situations is crucial. Environment puts various constraints on the project. For instance, slow perception property makes it tougher to perceive the exact result of the action taken, 16.67 ms time limit for processing frame data and deciding next input is forcing agents to decide fast. In addition to that, a fighting agent is designed to beat its opponents; therefore it should be mostly unpredictable, wise and nimble. On top of that, as skills differ for each character and there is a tournament for each one, an AI better be powerful in all available and not-yet-available characters. For simplicity, this project focused on only ZEN Standard league. Rule-based agents perform so satisfactorily but they seek a human expert to define the rules and their behaviors are generally so easy to predict. When the previous winners' codes are analyzed, a rule-based agent named Machete from 2015 competition is worth to mention. Machete applied very simple but effective rules in its decision mechanism. It was wise: its decisions are mainly based on the distance between players and energies of both; it was unpredictable: if no base conditions are met, it introduces randomness to complicate the prediction of its actions; it was nimble: it never stands still, increases chances by continuously moving. In the light of these, most of the contestants chose to stand on the shoulders of giants in 2016 and used Machete as their base logic. On the other hand, last year's applicants also applied several machine learning techniques to overcome the predictability issue and to increase their performance. I've decided to use machine learning on my AI to avoid the need of an expert of fighting games. This term, a winner-voting agent that asks for action advise of 2016 competition's best three agent is introduced. As the last term's GaussianAI's main problem, this winner-voting agent had mostly surpassed the given decision time, therefore a faster winner-voting agent was needed. When examined it was seen that even two of the winners had time problems, so it is normal to exceed the time limit with three winners deciding and voting one by one; recalling that multithreading is not allowed. In order to achieve a better agent, winners are modeled by neural networks. However, machine learning comes with some more challenges. In order to train the neural networks, I need enough data to build a good model that can generate successful predictions. In my case, there is no data readily available. Log collection and preprocessing the data is needed including feature selection, feature encoding and such challenging data preprocessing steps. Training, integrating and evaluating the AI are the other challenging parts of the project.

# Solution Approach

## Achievements for the 1st Term

### Random Agent Implementation

As the first step, I've implemented a random agent. It selects a random action and outputs the series of input keys for each frame using CommandCenter class [5].

### Data Collection

I've added a log writer to the random agent. It logs out remaining time, player's action, opponent's action, distance between players on the x-axis and the difference between current hit point and the hit point fifteen frames ago. Hit point difference calculation is based on my agent's fifteen frames delayed perception. In order to collect data, my agent RandAct_LogAI competed with previous years' winners multiple times. For simplicity, this small tournament was all on ZEN character. All the collected data is in comma-separated-value format. In order to minimize the size of the logs, only the crucial features are recorded.

LOG COLLECTION ALGORITHM

```
start
    Define DELAY_FRAME (Default is 15 frames).
    Prepare logger.
    for i = 0 : (n-1) all n frames
        Make a random action.
        Create a MyState object with current distance on X and Y axes, player's and opponent's actions and hit points.
        Put the state into hash map in a (remainingFrame – DELAY_FRAME, currState) key-value mapping.
        if ((remainingFrame) key exists in hash map)
            Compare current hit points with prevState's from ((remainingFrame), prevState).
            if (opponent's hit point difference != 0)
                Buffer (opp.'s hpDiff, player's act., opp.'s act., distanceOnX, distanceOnY) to logger.
            endif
            if (player's hit point difference != 0)
                Buffer (player's hpDiff, opp.'s act., player's act., distanceOnX, distanceOnY) to logger.
            endif
            Remove (remainingTime) key from hash map.
        endif
    endfor
    Save the buffered data in a comma-separated-value file.
end
```

### Dataset Selection

Having labeled game data led me to use regression from supervised learning. This term, Gaussian Process Regression (GPR) and Support Vector Machines (SVM) are used with Weka Data Mining Tool. On the other hand, there were many data sets that I've created and also a merged data file. Merged data was taking

too long to train; therefore I've decided not to use it this term. To decide on the dataset and see the model evaluations on both algorithms, an experiment on Weka is conducted.

GPR and SVM algorithms have run with nine distinct log files. As the evaluation type, ten-fold cross validation with ten repetitions is used. Radial Basis Function is selected as the kernel and in preprocess step, data is normalized for both GPR and SVM algorithms. Below are the settings for the experiment.

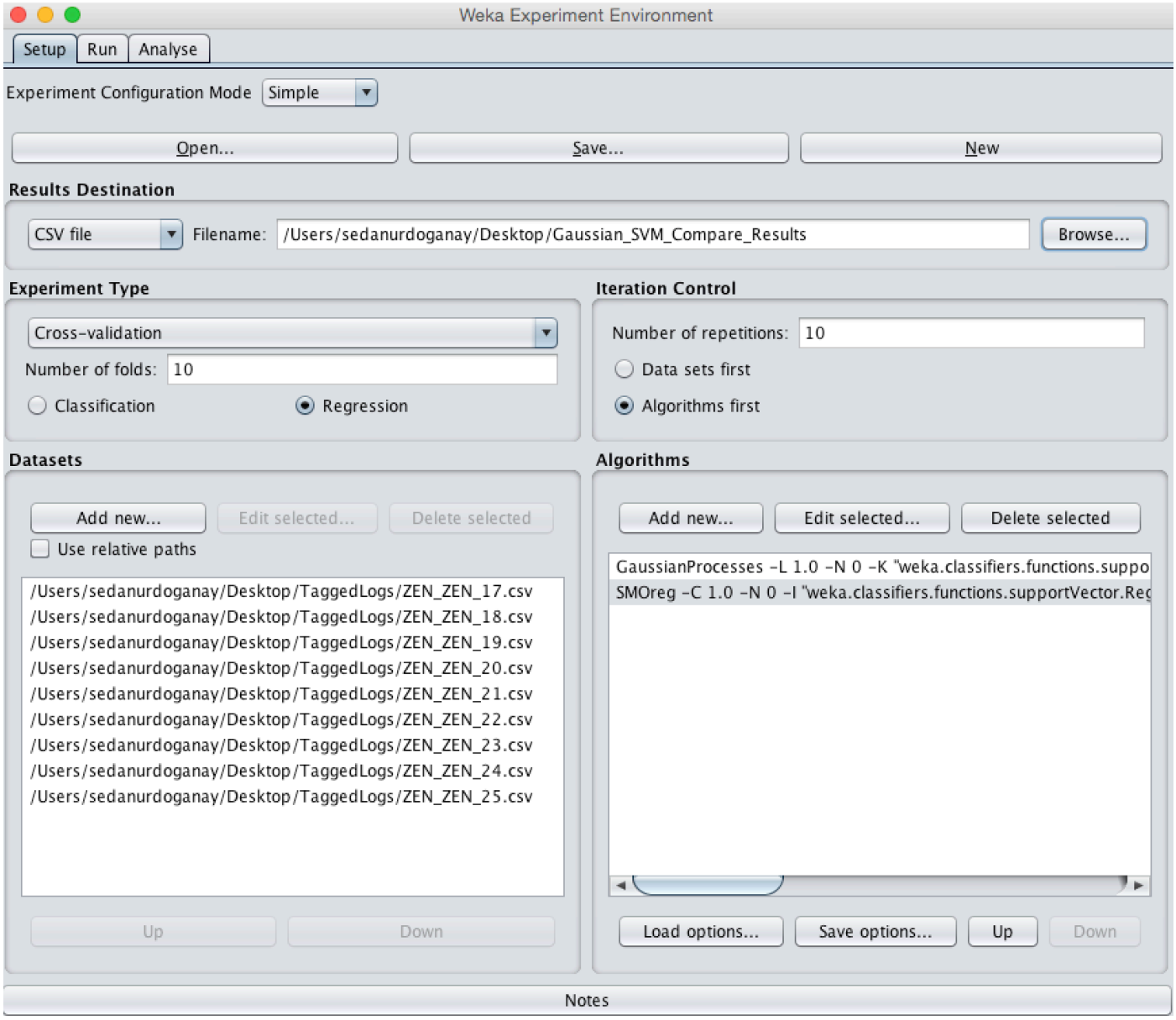



**Figure 7 - General Settings of the experiment**

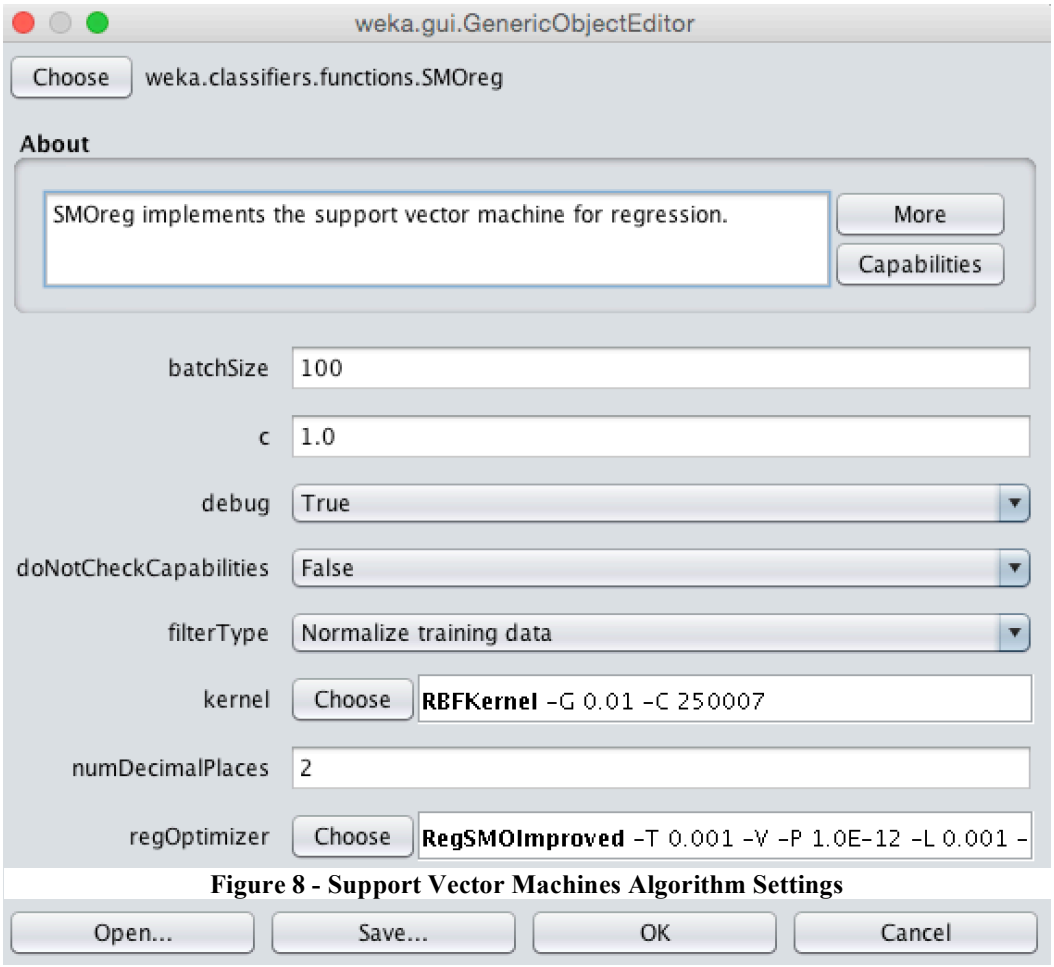



**Figure 8 - Support Vector Machines Algorithm Settings**

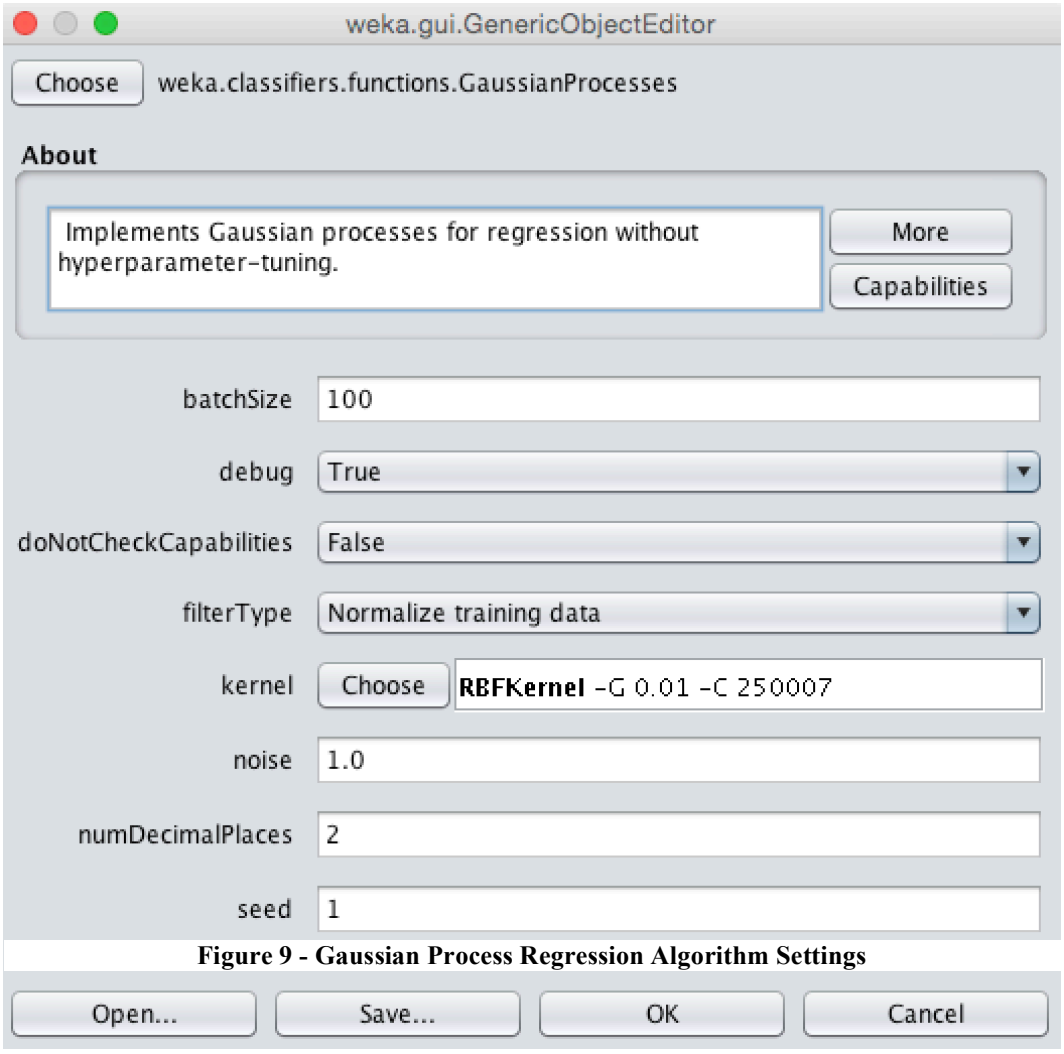



**Figure 9 - Gaussian Process Regression Algorithm Settings**

In the lights of the experiment results below, it is decided to use ZEN_ZEN_23.csv as training data. It has the lowest root mean square error with a high correlation coefficient on both algorithms.

**Test output**

```
Tester:     weka.experiment.PairedCorrectedTTester -G 4,5,6 -D 1 -R 2 -S 0.05 -result-matrix
Analysing:  Correlation_coefficient
Datasets:   9
Resultsets: 2
Confidence: 0.05 (two tailed)
Sorted by:  Root_mean_squared_error
Date:       1/3/17 2:46 PM


Dataset                   (2) functio | (1) func
------------------------------------------------
ZEN_ZEN_23               (100)   0.79 |   0.73 *
ZEN_ZEN_18               (100)   0.67 |   0.61 *
ZEN_ZEN_17               (100)   0.62 |   0.62
ZEN_ZEN_21               (100)   0.62 |   0.63
ZEN_ZEN_22               (100)   0.73 |   0.73
ZEN_ZEN_24               (100)   0.72 |   0.71
ZEN_ZEN_20               (100)   0.80 |   0.76 *
ZEN_ZEN_25               (100)   0.63 |   0.66 v
ZEN_ZEN_19               (100)   0.64 |   0.63
------------------------------------------------
                            (v/ /*) |  (1/5/3)


Key:
(1) functions.GaussianProcesses '-L 1.0 -N 0 -K \"functions.supportVector.RBFKernel -G 0.01 -
(2) functions.SMOreg '-C 1.0 -N 0 -I \"functions.supportVector.RegSMOImproved -T 0.001 -V -P
```

**Figure 10 - Experiment Results**

## Regression Implementation

I've implemented some Java programs; Csv2Arff that converts CSV files to ARFF files to preprocess my data format, RegressionModel that handles model creation and evaluation, RegressFromModel which loads a model and make predictions on individual instances. Weka API is used for implementation. First, converted all my data to Attribute-Relation File Format (ARFF) because it is easier to see the attribute properties in ARFF type as illustrated below.

```
@relation FileName

@attribute attr1 numeric
@attribute attr2 {LABEL_A,LABEL_B}

@data
10,LABEL_A
-2,LABEL_B
```

**Figure 11 - An example of Attribute-Relation File Format**

I've started with GPR therefore created a GPR object and fed it with ZEN_ZEN_23.arff to build and evaluate the model in RegressionModel. Below is the result.

```
Console:

***time it takes to load the dataset: 66ms
***time it takes to build the classifier: 2843ms
***time it takes to evaluate data: 20003ms

Gaussian Processes

Kernel used:
  RBF kernel: K(x,y) = e^-(0.01* <x-y,x-y>^2)

All values shown based on: Normalize training data

Average Target Value : 0.4232877696119481
Inverted Covariance Matrix:
    Lowest Value = -0.03612359957989765
    Highest Value = 0.9894581576984652
Inverted Covariance Matrix * Target-value Vector:
    Lowest Value = -0.4573956876209073
    Highest Value = 0.6517127779565395

Evaluation results:

Correlation coefficient                  0.5765
Mean absolute error                     25.0598
Root mean squared error                 31.1744
Relative absolute error                 86.5885 %
Root relative squared error             85.9603 %
Total Number of Instances             1129
```

**Figure 12 - Output of RegressionModel program for GPR**

Then I've tested a random state to make a hit point prediction with the loaded model and calculated the time performances in RegressFromModel. Below is the result.

```
Console:

***time it takes to load the model: 260ms
***time it takes to load the dataset: 16ms

***time it takes to classify the state: 15ms
prediction -9.34033628279195
actual 10
```

In agent initialization

In each frame process

**Figure 13 - Output of RegressFromModel program for GPR**

As seen, GPR is poor in time performance. When this model is integrated with the agent, some frames will have repetitive inputs because the agent won't conclude its input decision in dedicated time for one frame, 16.67 ms. Nevertheless, I'll integrate GPR to my agent.

Secondly, I've tried SVM regression using RBFKernel. It took 13 ms to classify a state that is nearly the same results with GPR. Therefore I've decided to use Linear Kernel to decrease the computation time. Below are the equivalents of previous results for Support Vector Machine Regression.

Console:

```
***time it takes to load the dataset: 56ms
***time it takes to build the classifier: 3708ms
***time it takes to evaluate data: 23690ms

SMOreg

Kernel used:
       Linear Kernel: K(x,y) = <x,y>

weights (not support vectors):
 +       0.1492 * (normalized) Attack=CROUCH_A
 -         0.0638 * (normalized) Attack=AIR
 -         ...
 -       0.0654 * (normalized) Attack=RISE
 -       0.1726 * (normalized) Attack=AIR_DB
 -       0.0317 * (normalized) CounterAttack=JUMP
 -       0.1405 * (normalized) CounterAttack=AIR_UA
 +       ...
 -       0.0337 * (normalized) CounterAttack=RISE
 +       0.1832 * (normalized) CounterAttack=AIR_DB
 +       0.0019 * (normalized) distanceOnX
 -       0.0014 * (normalized) distanceOnY
 +       0.4766

Number of kernel evaluations: 637885 (99.036% cached)
Evaluation results:

Correlation coefficient                  0.6325
Mean absolute error                     18.4013
Root mean squared error                 30.2324
Relative absolute error                 63.5815 %
Root relative squared error             83.3628 %
Total Number of Instances             1129
```

**Figure 14 - Output of RegressionModel program for SVM**

Console:

```
***time it takes to load the model: 174ms
***time it takes to load the dataset: 20ms

***time it takes to classify the state: 6ms
prediction -43.772711023122845
actual 10
```

**Figure 15 - Output of RegressFromModel program for SVM**

**Model Integration**

I've implemented GaussianAI on top of the RandomActionAI. The saved regression model from previous programs copied to /ai/aiData/GaussianAI in FightingICE platform. Regression model and an empty dataset in ARFF are uploaded in initialization method. Empty dataset is used for defining the attribute relations for each state. In addition to model and dataset, an array of possible actions of the character is created at the beginning. In processing method, I am implementing my own decision algorithm using the ML model. Below is my decision algorithm for deciding the next action using current frame data.

```
private String calculateNextAction() {
	double maxHP = 0;
	int actionIndex = 0;
	for (int i = 0; i < actions.length; i++) {
		double hp = predictHPDiff(actions[i]);
		if (hp > maxHP) {
			maxHP = hp;
			actionIndex = i;
		}
	}
	return actions[actionIndex];
}
```

**Figure 16 - Algorithm to decide on the next action**

Main idea is to predict the hit point change for each action I could take and then choose the one with the maximum hit point gain. The predictHPDiff method used above creates a State object with current frame data and given action, then it calls calculateHPDiff. Below is the calculateHPDiff method of the agent.

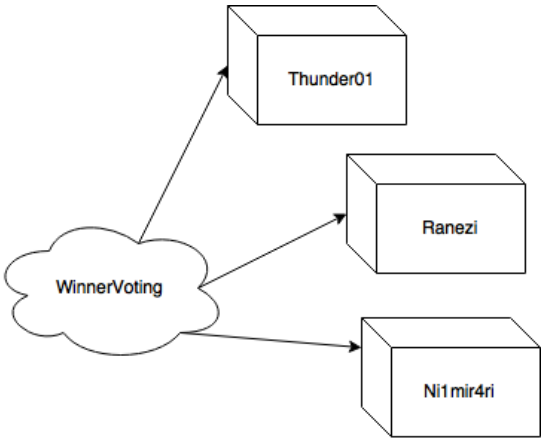
```
public double calculateHPDiff() {
		double[] values = new double[dataset.numAttributes()];
		values[0]=0;
		values[1] = dataset.attribute(2).indexOfValue(myAction);
		values[2] = dataset.attribute(2).indexOfValue(oppAction);
		values[3] = distanceOnX;
		values[4] = distanceOnY;
		Instance inst = new DenseInstance(1.0, values);

		double predGauss = 0;
		try {
			predGauss = gaussianModel.classifyInstance(inst);
		} catch (Exception e) {
			e.printStackTrace();
		}

		System.out.println("prediction " + predGauss);
		hpDiff = predGauss;
		return predGauss;
	}
```

It is effortless to change the model to SVM regression via this decision algorithm; changing the model file in FightingICE platform is enough.

## Achievements for the 2nd Term

### Winner-Voting Agent Implementation

In order to have a stronger agent than the winners, a winner-voting agent is implemented. The agent asks votes from the best three agents except Machete, as will be called advisor agents, when each new frame arrives and takes the most voted action. The advisor agents are Thunder01, Ranezi and Ni1mir4ri. If ties exist in voting, winner-voting basically takes the action which Thunder01, the best agent, recommended. Despite Machete is the second best agent, it is excepted from this three-agent advisory group, because it is a rule-based agent that most of the powerful agents implemented their logic on top of its. For instance, Thunder01 uses Monte-Carlo Tree Search on top of Machete's logic, so that in a sense Machete's recommendation is already included.




### Running a League

The platform has a round robin tournament argument to pass but it didn't work properly because not each agent indicates its character as ZEN. Therefore I've written my own round robin tournament system with python. Below is the round robin logic for starting a tournament with .jar agent files.

```
def start_game():
  source = "path-to-source/FTG3.10/data/ai"
  agents = list(map(lambda f: f[:-4], filter(lambda filename: filename.endswith(".jar"), os.listdir(source))))
  for agent1 in agents:
    for agent2 in agents:
      for i in range(GAME_NUM):
        print("Starting the game {0} between A1={1} and A2={2}".format(i+1,agent1,agent2))
        game = manager.createGame("ZEN", "ZEN", agent1, agent2)
        manager.runGame(game)
        print("Game finished")
        print ("-"*20)
  sys.stdout.flush()
  print ("-"*20)
  print("all games finished")
```

After running the FightingICE platform in order to run with python, it starts to listen the decided port for game to start. The command below calls run_round_robin.py, which uses the logic above. It runs five games for each match-up.

```
$ python run_round_robin.py -n 5
```

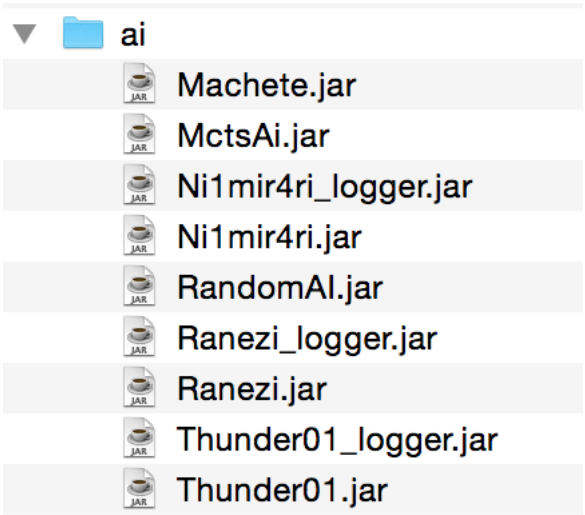
**WinnerVoting Agent Results**

By virtue of multi-threading prohibition in the game environment, winner-voting agent asked and got votes from each advisor agent one by one. Thunder01 and Ranezi were each taking 16 to 40 milliseconds to process whereas Ni1mir4ri was taking only up to 2 milliseconds. All these process times combined with the linear flow of the voting system create a big timing problem in winner-voting agent. As one of the most important constraints the game environment has, deciding on the action in 16 milliseconds limitation cannot be met. Below is the result of a ZEN Standard League ran and evaluated in organizer's scoring system to see winner-voting's performance.

| Agents | Total Victories | |
|---|---|---|
| Thunder01 | 11 | |
| Machete | 10 | |
| Ni1mir4ri | 8 | |
| Ranezi | 6 | |
| WinnerVotingAI | 4 | *Time problem |
| RandomAI | 2 | |
| GaussianAI | 1 | *Time problem |

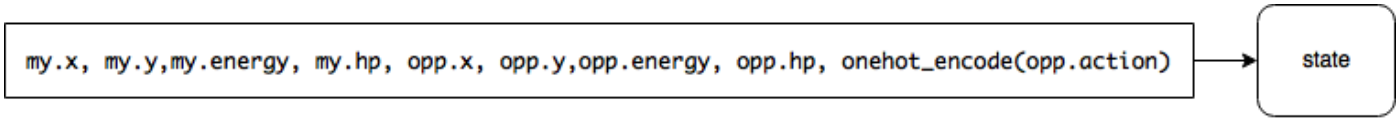
**WinnersModel Agent – Data Collection**

In order to overcome the process time limitation, I decided to model winners in neural networks and use these networks to vote. In this manner I've collected data by adding log collection part to winners' code and ran a tournament with five games for each match-up. Below is the list of agents involved in the tournament. Nearly 13 MB log collected for each agent, which is in total 38.8 MB.

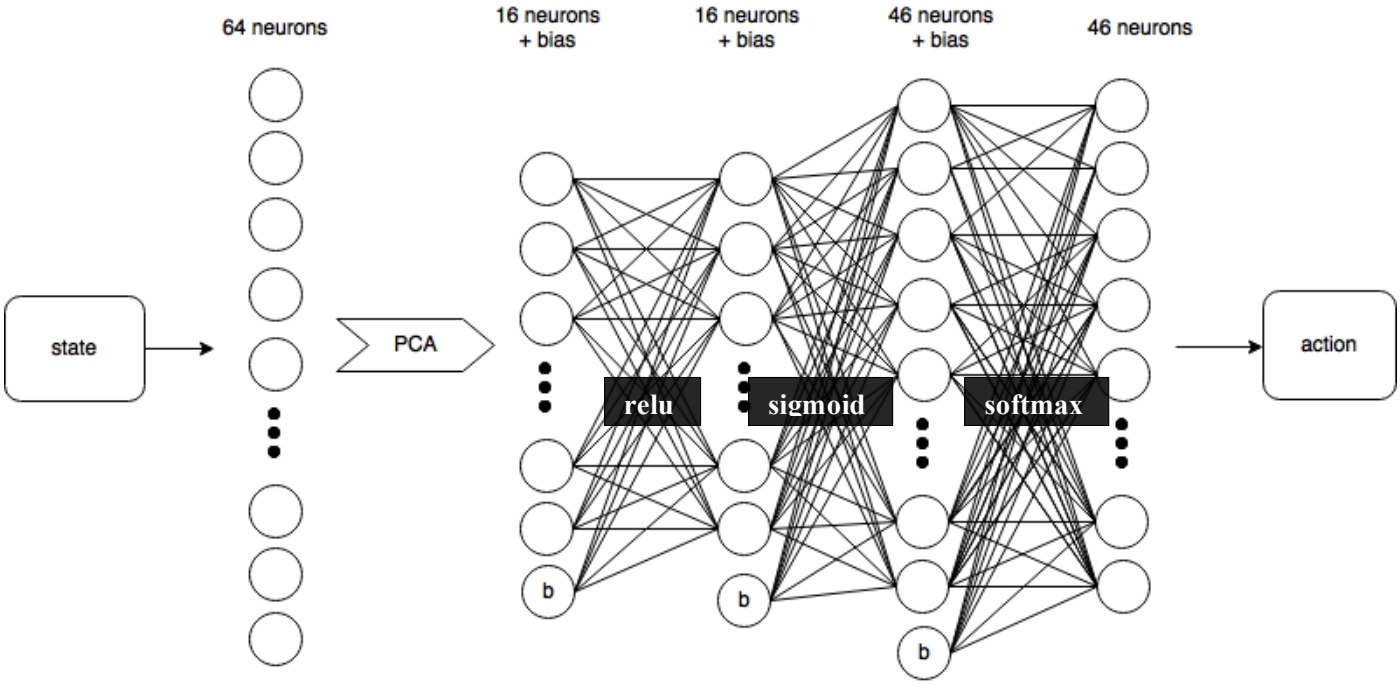


**WinnersModel Agent – Neural Network Training**

First the program reads the logs of the target agent with Pandas and dividing them into train (70%), test (10%), and validation (20%) parts. The data is preprocessed in this part. The states with the agent took Recovery or Throw actions are eliminated because these actions are not taken consciously, the game environment forces them as the opponent hits the agent. The nominal values, actions, are turned to their binary representations in this part. Because we eliminated some of the available actions our outputs, Agent's actions, are onehot encoded with respect to the action set without forced ones. On the other hand, the opponent's actions are encoded with respect to all possible actions of the ZEN character's motions, in order to avoid the situation of not having some actions in dataset and have problems in encoding when opponent does them in the game.



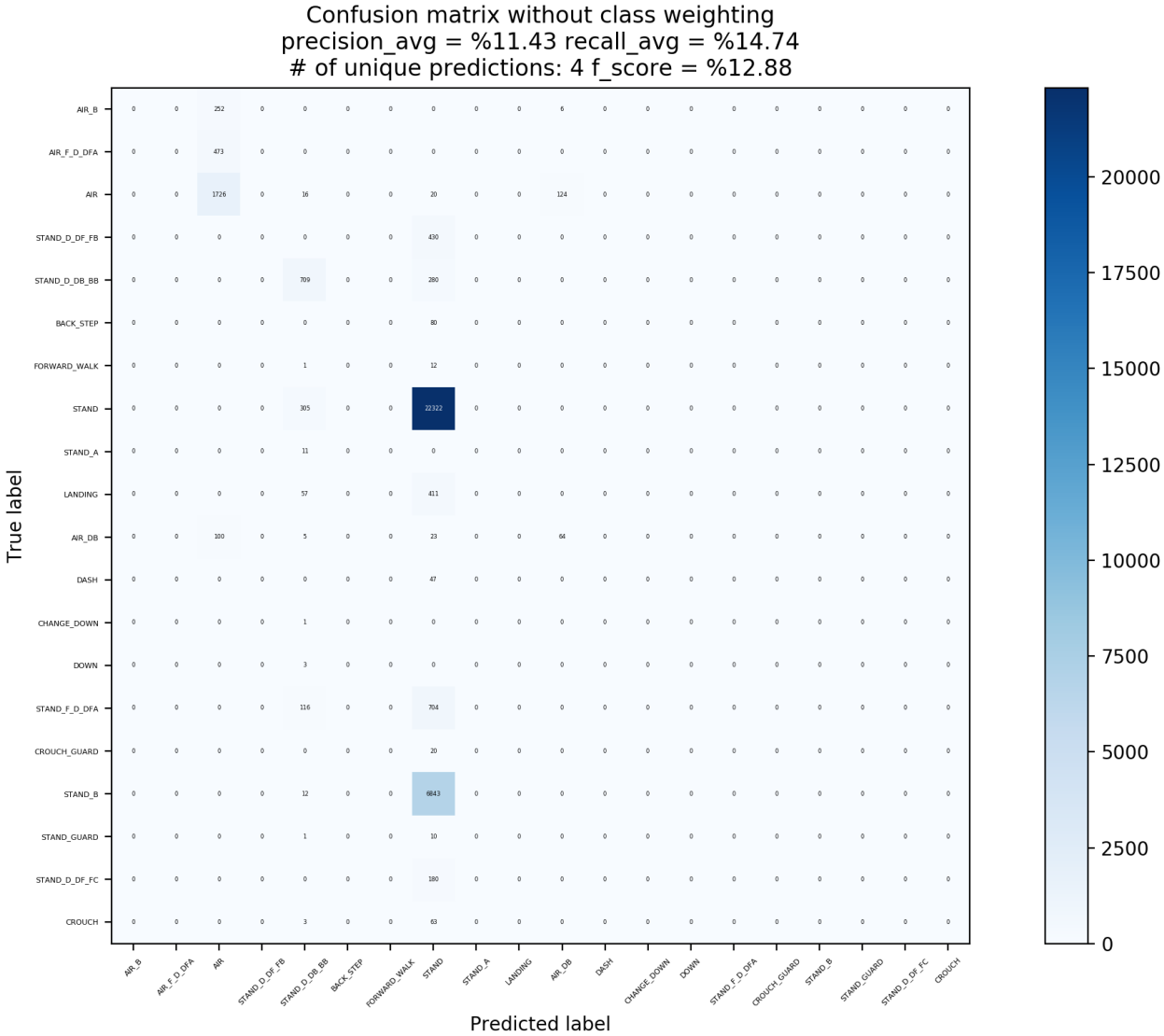

After the preprocess on our data, the input's dimension becomes 64 and the output becomes 46. Before reducing the dimension using Principal Component Analysis (PCA), it was surpassing the decision time limit even with one agent's model. Applying PCA onto our input and computing 16 principal components to work on helped in decreasing the process time from nearly 32 ms to 2 ms without much data loss. During deciding the hyper-parameters such as number of neurons in each layer, learning rate the validation data is evaluated and taken action accordingly.

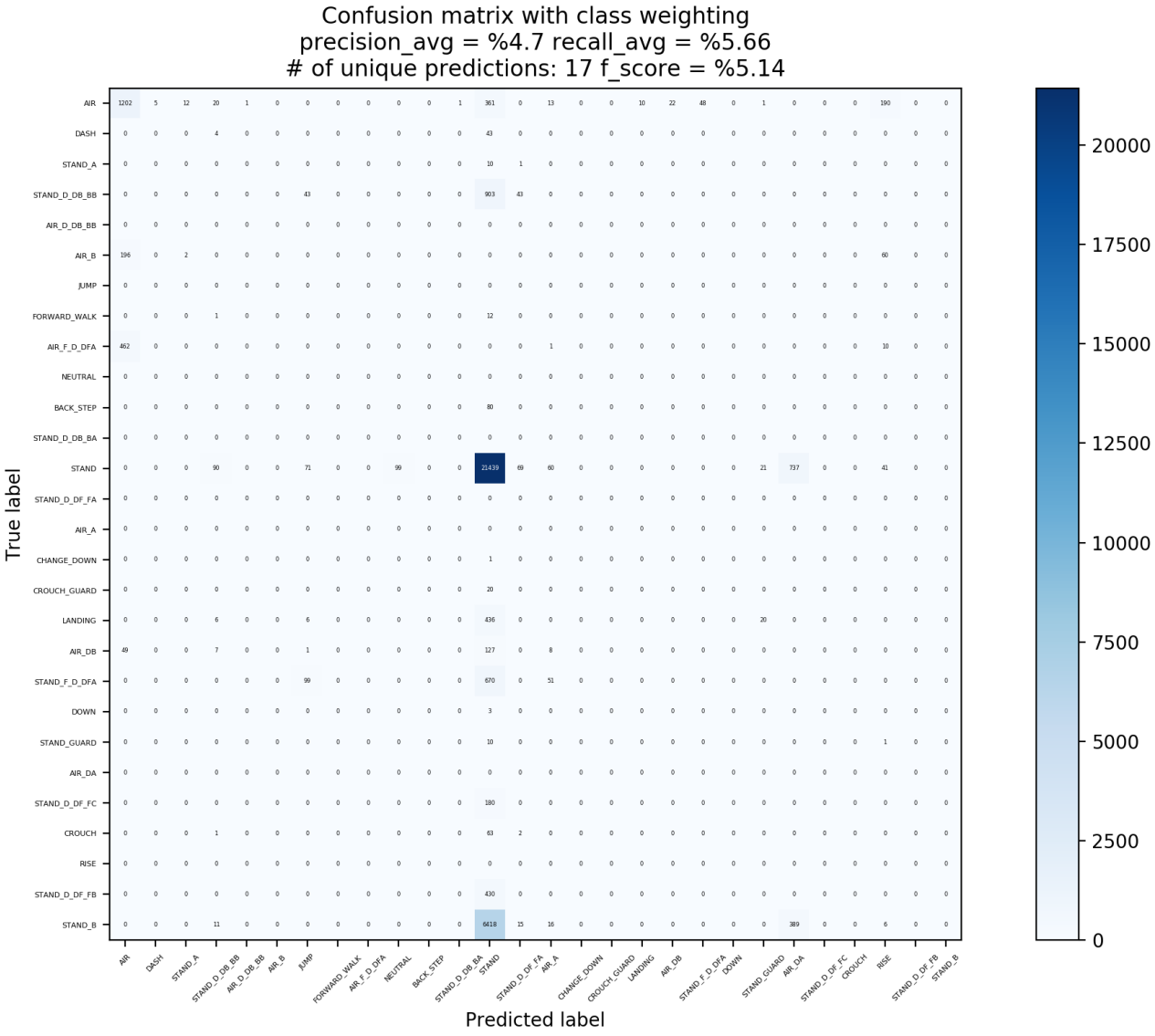


The data collected was extremely imbalanced. It was consisting too much STAND and AIR actions. To illustrate, by recommending only these two actions the accuracy reaches around 50-60 percent. In order to defeat the side effects of an imbalanced dataset, class weighting is used. Weights of each action class calculated as counting the number of occurrences of an action, term frequency (TF), and taking reverse of it. The formula can be seen below.

$$w_i = (TF_i + 1)^{-1}$$

This class weighting added in the last part of the network. The logits that the network concluded are matrix multiplied with the class weights and this value is passed to loss function rather than only the logits itself. In this way, highly populated classes will be penalized, while the unfortunate less occurred actions are supported. To illustrate, below confusion matrix is used.






## WinnersModel Agent – Neural Models Integration

After training all the networks, Tensorflow models' checkpoint files are saved and PCA's are pickled. Given checkpoint files and PCA files, our WinnersModelAI's Tensorflow graphs of each model are restored and session is started. In each frame our agent concatenates the information about the state and forwards it to agent neural models. They apply restored PCAs on it and feed the network and give action outputs. After that given outputs are voted and most voted one is taken.

## Results and Discussion

In the first parts of the project, I've started with learning about machine learning and artificial intelligence. Without any prior knowledge it was hard to understand and implement machine-learning algorithms. Therefore I've used Weka API to implement ML algorithms and studied related topics to get to know what I am implementing. Collecting the data without considering the 15 frames delay creates a misconception of the hit point change. I've stumbled into this mistake at the beginning and corrected by calculating the hit point difference after fifteen frames. My Gaussian model's accuracy was 31.1744 root mean squared error. The accuracy could be increased with feed the model with bigger data. On the other hand, the prediction time took too much. Even if the accuracy of the model were high, in a real time fighting game it doesn't matter. When the agent cannot decide on the action in time and it gets attacked while thinking. This would have been solved via using less complicated algorithms and minimizing the number of features. Shortly, GaussianAI underperformed. The main focus for this term was to meet the constraints. WinnerVoting agent was a simple and effective idea with a side effect of mixing up strategies of individually great agents. They work well alone but may not work well together. Unfortunately, we couldn't analyze this side effect because our agent was surpassing the time limit. Therefore we developed a fast voting agent with Artificial Neural Networks. Their modelings of the agents were not that good because of imbalanced and small dataset. Also the initializing time of the WinnersModel was surpassing the initializing time limit, 5 ms, by 300 ms. When competing with powerful agents, this problem makes the agent an open and defenseless target at the beginning of each game. As can be seen in the results WinnersModel is performing better than the WinnerVotingAI but cannot pass the best agents.

| Agents | Total Victories | |
|---|---|---|
| Thunder01 | 13 | |
| Machete | 12 | |
| Ni1mir4ri | 10 | |
| Ranezi | 8 | |
| WinnersModel | 6 | |
| WinnerVotingAI | 4 | *Time problem |
| RandomAI | 2 | |
| GaussianAI | 1 | *Time problem |

**Figure 17 - Game results**

# Conclusion and Future Work

This work contributes to the artificial intelligence in games but as some games can be seen as simulations of the real world, it can be said that the agent I am going to build can be used in a robot brain too. With the dawn of big data and artificial intelligences that process it very quickly by the means of effective algorithms and faster CPUs, game sector will be using more and more machine learning in the field.

First term, I've learned about machine learning, data mining, and artificial intelligence. I've examined the framework, previous agents, some related works, generated some game data, work on some machine learning algorithms such as Gaussian Process Regression and Support Vector Machines Regression and also implemented an AI agent using Gaussian Process Regression.

Second term, I've examined the new features of the game environment and made additions to it such as organizing a tournament from Python. After developing a Java-based winner-voting agent it was seen that time limit to give the action decision in a frame is really crucial and it was withholding our agent to actually perform its decisions in the right time. Therefore instead of simply going through three agents very time consuming processes, I decided to speed up the overall decision making by taking advantage of Artificial Neural Networks. I've collected some data for the training with the help of modified agents and implemented Artificial Neural Networks on top of it. During the training process, I've had problems with the data being imbalanced and having inadequate diversity. Therefore class weighting is used. On the other hand the time to complete an iteration was not as fast as we need, to improve the timing Principal component analysis is applied and 64-feature data is simplified to 16-column.

Training time can be huge with even small size datasets. On the other hand with more data, our models become more accurate. In Gaussian process regression, a 32 KB log file was used for training and evaluating. In Artificial Neural Networks a total of 50 MB log folder containing three agents' logs, with each is nearly 16 MB, is used for training and evaluating. A bigger dataset with multi-process training would become more powerful than these trainings.

Weka and Tensorflow are used for their easiness to implement machine learning algortihms. Tensorflow was more engaging in terms of coding by hand rather than interacting with a graphical user interface. A great challenge would be implementing a deep reinforcement agent without using any libraries.

# References


[1] - Fighting Game AI Competition.(2016) [Online].
Available: http://www.ice.ci.ritsumei.ac.jp/~ftgaic/index.htm

[2] - Lee Sedol. (2016, Mar 16 ). “Google Deep Mind’s AlphaGo: How It Works” [Online]. Available: https://www.tastehit.com/blog/google-deepmind-alphago-how-it-works/

[3] - Machine Learning Group at the University of Waikato. (2016). Weka 3: Data Mining Software in Java [Online]. Available:
http://weka.sourceforge.net <http://weka.sourceforge.net/>

[4] – Introduction of the relevant data and methods. (2016) [Online]. Available:
http://www.ice.ci.ritsumei.ac.jp/~ftgaic/Downloadfiles/Introduction%20of%20the%20relevant%20data%20and%20methods.pdf

[5] - Introduction of the CommandCenter Class. (2016) [Online]. Available:
http://www.ice.ci.ritsumei.ac.jp/~ftgaic/Downloadfiles/Introduction%20of%20the%20CommandCenter%20class.pdf <%22>

[6] - Results of the Game. (2016) [Online]. Available:
http://www.ice.ci.ritsumei.ac.jp/~ftgaic/index-3.html

[7] - Asayama, Kazuki et. al. “Prediction as Faster Perception in a Real-time Fighting Video Game” The IEEE Conference on Computational Intelligence and Games, pp 517-522, August 31, 2015

[8] - Input Conversion Table. (2016) [Online]. Available:
http://www.ice.ci.ritsumei.ac.jp/~ftgaic/Downloadfiles/Brief%20table%20of%20ZEN's%20skills.pdf <%22>

[9] - Stefan Kramer. Gaussian Process for Regression. [Online]. Available:
http://old.opentox.org/dev/documentation/components/gaussianregressions

[10] - Ron Kohavi; Foster Provost (1998). “Glossary of terms”. Machine Learning. 30: 271-274

[11] - Lueangrueangroj, Sarayut and Kotrajaras, Vishnu. (2009) “Real-Time Imitation Based Learning for Commercial Fighting Games” [Online]
https://www.cp.eng.chula.ac.th/~vishnu/gameProg/papers/CGAT_Real%20Time%20Imitaion%20Based%20Learning.pdf

[12] - - Asayama, Kazuki et. al. “Prediction as Faster Perception in a Real-time Fighting Video Game” The IEEE Conference on Computational Intelligence and Games, pp 517-522, August 31, 2015

[13] - Yuto Nakagawa and Kaito Yamamoto et. al. “Predicting the Opponent's Action Using the k-Nearest Neighbour Algorithm and a Substring Tree Structure” The IEEE 4th Global Conference on Consumer Electronics, pp 533-534, 2015

[14] - Joson Brownlee. (2016, July 5). “How to Normalize and Standardise Your Machine Learning Data in Weka” [Online]. Available: http://machinelearningmastery.com/normalize-standardize-machine-learning-data-weka/
[15] - Salzahrani. (2016, Dec 31). Radial Basis Function Kernel [Online].
Available: https://en.wikipedia.org/wiki/Radial_basis_function_kernel
[16] -Scott Fortmann Roe. (2012, May). “Accurately Measuring Model Prediction Error” [Online].
Available: http://scott.fortmann-roe.com/docs/MeasuringError.html