SELF-DRIVE MARIO KART AI

Jianhuan Zeng (jz2883) *Haikun Du (hd2377)* *Junyi Chen (jc4805)*

Columbia University Columbia University Columbia University

ABSTRACT

In this project, we developed an AI system to play Mario Kart 64 automatically. Our system has two main modules. First part is a kart moving action prediction module. A search AI generates a training set associating screenshots along with the best action. Then Nvidia autopilot model as the real-time CNN to recognize road features. The second part is an item usage module. It first recognizes what kind of items the character obtains. Then the AI strategy will decide when to use such items. Video demo of this project is available at https://www.youtube.com/watch?v=YtORFeG191s.

***Index Terms*—** Mario Kart, Autopilot Model, YOLO

1. INTRODUCTION

Mario Kart as a popular kart racing game introduces interesting challenges: racers need to navigate hazards as well as use items effectively. In this project, we want to develop a system that can play this kart racing game while only based on the information on the screen. Models developed before on Mario Kart AI only predict steering action, no advanced techniques such as drifting and using items at opportune times. Also, most of the self-driving models depend on human action. They extract features from pre-recorded human playing videos. LSTM network is also used in such systems to extract time features. We can divide the system into two separate parts: self- driving module and item usage model. The first part in charge of choosing the best route as well as drifting. This part is quite similar to an autonomous driving problem. By using deep learning methods specifically convolutional neural networks, we can extract features from the screen through screenshots of each frame. The second module is about using items. In Mario Kart 64, there are 14 different items and different items have a different effect. To simplify the problem, we group all 14 items into three groups. To use items efficiently, we make a playing strategy for each group. Since some items in Mario Kart games are related to other game players or robots. We also need to detect the position of other karts while playing the game. Our experiment on both tracks (GP and TT. Grand Prix mode includes items and other 7 robots. Time-Trial mode only cares about time.) shows that the AI yields competitive performance. It only uses mushrooms (speed up items) when the kart drives on straight roads and only use three shells (defending items) when the system detects there are other karts around us.

2. RELATED WORK

2.1. Nvidia Autopilot Model

Nvidia Autopilot Model [1] is a breakthrough model for real-time autonomous driving in real life, which is convolutional neural networks (CNNs) to map the raw pixels from a front-facing camera to the steering commands for a self-driving car. This powerful end-to-end approach means that with minimum training data from humans, the system learns to steer.

Since Mario Kart 64 represents a simplification of real-life autonomous driving, we would like to apply the Autopilot Model to our gaming AI.

2.2. TensorKart

TensorKart [2] is the very first model to develop self-play Mario Kart AI, which makes use of imitation learning. Real-time deep learning controllers are often trained using imitation learning. In TensorKart, an expert is recorded performing the task, and observations and related actions are recorded at each time-step. A neural network is then trained using these recordings as a dataset, thus learning to imitate the expert.

However, imitation learning controllers depend on the human level standard. And sometimes experts are too good, thus the controller never learns to correct itself or recover, and small errors in prediction accumulate over time.

2.3. NeuralKart

NeualKart [3] is another Mario Kart AI, which raised the search method. And the project yields competitive performance to human players.

However, it only determines the best steering action to take from a given game state, no advanced techniques such as drifting. And the models developed did bad at turning, kart reacts slowly especially at large turning since the time features are not taken into consideration.

3. METHOD

3.1. Emulator

After comparing various emulators such as Mupen64plus and Sixtyforce, we choose Bizhawk as our emulator due to its stable performance and incredible features.

Bizhawk is capable for Lua scripting, which enables the emulator to take a screenshot, check memory status, save states, set joystick actions, and test specific circumstances automatically.

3.2. Search - Steering & Drift

The motivation for doing the search to create dataset is to replace imitation learning and save human efforts.

The kart movement basically consists of steering angle, drift and accelerate actions. Here we set acceleration key is pressed as default since kart is only able to move in this condition. Thus, we need to estimate the steering angle of the kart and whether to drift or not.

The N64 joystick originally return signed bytes from -128 to 127 as steering angles, and we remapped them into [-1,1]. We then divide steering angles into 11 actions: -1, -0.8, -0.6, -0.4, -0.2, 0, +0.2, +0.4, +0.6, +0.8, +1. And drift actions has two options: 0 (release key) and 1 (press key). Initially, the search space contains 21 actions, each drift action has a corresponding steering angle except for 0, since the kart will go straight no matter the drift key is pressed or not. Several actions have very few data, especially action pairs (0, +0.8), (0, -0.8), (0, +0.6), (0,-0.6), because the kart tends to drift at sharp turning, thus the dataset is skew, which leads to very bad prediction. Therefore we reduce the search space from 21 actions to 15 actions. The action mapping table shows in Table 1.

The search process runs offline, using the Bizhawk emulator to simulate different actions. During a search, the emulator saves the current position as the root state. It then tries 15 different actions from this root state and simulates the results for 30 frames.

After the simulation, a reward will be given to each action, the reward is defined as a weighted sum of the game process and current kart speed ($0.9 * game\ process + 0.1 * kart\ speed$), both of the values can be read from the game memory. Then, the action which gains the maximum reward along with the root state image will be saved in the dataset.

3.3. Sequential CNN Model

The Nvidia Autopilot Model Structure, as shown in Figure 1, is a popular and empirically good CNN model in self-driving. We also use the Nvidia autopilot model and modified it for our Mario Kart AI.

There are 5 convolution layers, and each layer is equipped with batch-normalization and ReLu activation. Then 4 fully connected layers are following and finally connected to the output layer. The differences are that we use convolution 3D for the first 2 convolution layers for our sequential inputs. Besides, our output is the probabilities of 15 classes, corresponding to the action mapping in the previous search section.

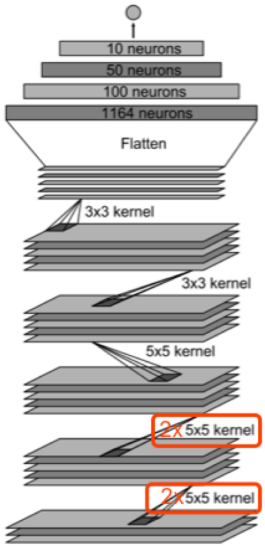

Fig. 1: Sequential CNN Model

As for sequential inputs, if we only train on a single screenshot, we could not capture features over time. Hence, we add 3 prior screenshots to the input. Therefore, we use convolution 3D layers with 5D tensor input (samples, timestep, frame hight, frame weight, channels).

In addition, the corresponding predicted steering output is also influenced by previous outputs with a discount factor γ. The predicted steering value in the i-th step is computed by $y_i = \gamma^3 * y_{i-3} + \gamma^2 * y_{i-2} + \gamma * y_{i-1} + y_i$.

3.4. AI Strategy of Item usage

Item usage plays a very important role in Mario Kart games. To simplify the problem, based on the effect of items, we first group all 14 items into three groups: instant items, angle-based items, and race-based items. During the game, the system will first to detect what kind of items we get. The instant group has eight different items. The AI will use such items immediately once we get them since we will get benefits at any time. For the other two groups, the system will hold the items then the status checking part starts to work. It only uses the item when the status variable (road type, competitors position) fits the requirement. Fig.2 shows the structure of our AI strategy.

Angle based group contains three different items: mushroom, three mushrooms, and gold mushroom. All three items are accelerators. For the item in this group, the AI will only use item when the kart drives on the straight road. We compute the slope of historical steering data as a criterion

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Drift	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
Steering	0	-0.4	-0.2	+0.2	+0.4	-1	-0.8	-0.6	-0.4	-0.2	+0.2	+0.4	+0.6	+0.8	+1

Table 1: Action Mapping Table

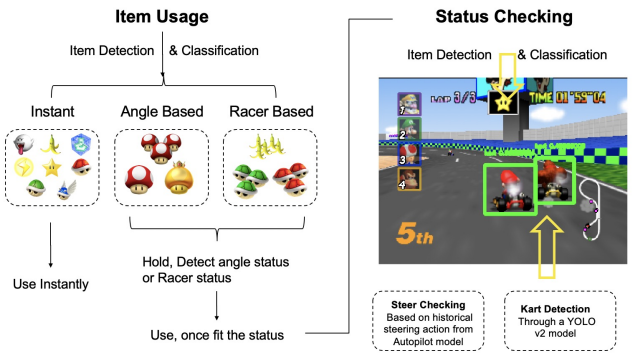

Fig. 2: Structure of AI Strategy

of straight road. Items in the race-based group are mainly defending items. It works only if there are other competitors around us. To get the position of competitors, we trained a YOLO V2 model on our own dataset. The details about this model will be discussed in the object detection section.

So, AI first use a decision tree model to classify the item we obtain belongs to which group. Then based on the strategy we have stated above, the system will decide when to use the items.

3.5. Object Detection

In the previous section, we have mentioned that we trained a YOLO V2 model on our own dataset to detect if there are other competitors around us. For the darknet model, we remove the last convolution layer. And adds three 3 * 3 convolutional layers with 1024 filters followed by an output layer. At first, we label the data for two classes: self and target. However, the performance of the model is not good enough because our own characters and competitors look very similar. And it does not work if we change the characters. Finally, we decide to detect all the karts. We know that our own kart always locates at the center of the screen. So, the kart which located at the center is player and others are competitors.

3.6. System Optimization

Motivation: During our experiment, we find that the delay of the network between the emulator and prediction model has a huge impact on the performance of the driving system.

Solution: Use a faster model for the driving system and add an extra step in AI strategy. For the driving, we also try to fine-tune a ResNet model to improve the performance of the drifting module. However, the delay has a huge impact on the steering prediction model such that the whole can only work for a few frames. On the other hand, the object detection model faces the same problem. Considering the stability of the whole system, we decided to choose Nvidia autopilot model as our self-driving model. And for the item-usage part, we add an edge detection part before the item classification model. Then YOLO V2 predicts only if we hold a racer based item.

Track Mode: TT	Baseline (s)	Human Time (s)	Mario Kart AI (s)
Moo Moo Farm	97.46	94.07	95.47
Luigi Raceway	129.09	125.30	125.39

Table 2: Game-play result of Self-Drive Mario Kart AI in Time Trial Mode

4. RESULT & EXPERIMENTS

4.1. Sequential CNN Model

We run our sequential Autopilot Model on Rainbow Raceway in Time Trial mode to test the effect of sequential inputs. We didn't test on simple tracks like Moo Moo Farm or Luigi Race Track since single-frame data already do well.

We run 5 races in Rainbow Raceway for each model, the original autopilot model with a single frame and the sequential model. The average sequential autopilot time is 385s. Although still less than human time (366s), it is a little faster than that of the original model (390s) [3].

4.2. Steering & Drift

We mainly collect dataset for Luigi Race Track and Moo Moo Farm in both Time Trial and Grand Prix Mode. We have around 6000 images for each track. As shown in Figure 3, the action distribution is skew, thus we reverse the image except for action with index 0 to make dataset more balance.

For the experiment, we set the result of NeuralKart[3] which only predict steering angle as a baseline. We compare with baseline and human of the time to finish the race which contains three full loops of the track without using items, the result is shown in Table 2. From the result, we can see that the game results are better when adding drift movement than the previous model, and are close to human play time.

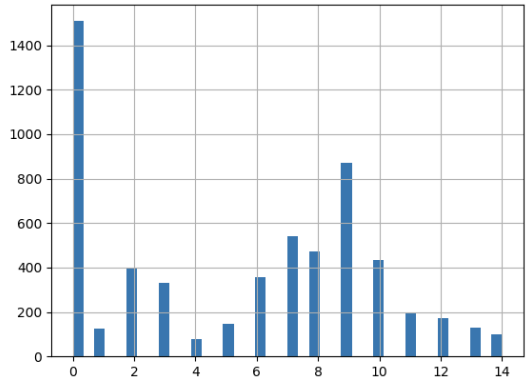

Fig. 3: Action Distribution of Luigi Race Track

Track Mode: GP	50cc (Rank)	100cc (Rank)	150cc (Rank)
Moo Moo Farm	1st	1st	1st
Luigi Raceway	1st	1st	3rd

Table 3: Game-play result of Self-Drive Mario Kart AI in Grand Prix Mode

4.3. Object Detection and Races with other robots

We labeled 800 screenshots (400 for Moo Moo Farm and 400 for Luigi Raceway) to train our model and randomly choose 700 pictures as the training set, others as the validation set.

For the experiment, we use AI to race with other robots in 50cc (easy mode), 100cc (medium mode) and 150cc (hard mode) on Moo Moo Farm track and Luigi Raceway track. Table.3 shows the result of final ranks.

5. CONCLUSION

We develop a self-driving Mario Kart AI with drifting and using items. The self-driving Mario Kart AI consist of two parts: the driving AI of steering and drifting control and the item-using AI for using various items.

Fig. 4: Kart Detection

The Mario driving AI is trained by a sequential autopilot CNN model and the training dataset is generated by a Search AI instead of imitation learning. The item-using AI includes 3 parts: items detection by Decision Tree, Kart detection by Yolo[4] and rule-based AI strategy of item usage.

6. FUTURE IDEAS

There are some awesome improvements could be made in the future. First, our Mario Kart AI could perform well as a human in most times, but when stuck on walls, the AI does not know how to fix. We may make full use of minimap to help.

Fig. 5: Illustration of Future Improvements

Moreover, our driving AI sometimes would prefer to drive on the grass. We could couple with Reinforcement Learning that rewards better performance and punishes error conditions to improve. In addition, our driving AI sometimes will jump when want to drift. The jump/drift misunderstanding may result from insufficient training.

Besides, the item-using AI now can only correctly recognize the enemies nearby, we could improve to enable target the enemies in the far front in the future work.

7. REFERENCES

[1] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[2] Kevin Hughes. Tensorkart: self-driving mariokart with tensorflow, 2016.

[3] Harrison Ho, Varun Ramesh, and Eduardo Torres Montano. Neuralkart: A real-time mario kart 64 ai.

[4] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016.