Supplementary

Edward Kim[1], Divya Gopinath[2], Corina Pasareanu[2], and Sanjit A. Seshia[1]

[1]University of California, Berkeley
[2]NASA AMES Research Center

1. Scenario Descriptions

For the experiment, we wrote four different scenarios in SCENIC language. The corresponding SCENIC programs are shown in Figure 1 and 2.

Scenario 1 describes the situation where a car is illegally intruding over a white striped traffic island at the entrance of an elevated highway. This scenario captures scenes where a car abruptly switches into or away from the elevated highway by crossing over the traffic island. The images from Scenario 1 are shown in Figure 3.

Scenario 2 describes two-car scenario where a car occludes the ego car's view of another car at a T-junction intersection . In the SCENIC program, to cause occlusion in scenes, we place an agent0 car within a visible region from ego car. Then, we place agent1 car within a close vicinity, defined by small horizontal (i.e. leftRight) and vertical (i.e. frontback) perturbations in the program, to agent0 car. The metric of these perturbations are in meters. The images from Scenario 2 are shown in Figure 4.

Scenario 3 describes scenes where three cars are merging into the ego car's lane. The location for Scenario 3 is carefully chosen such that the sun rises in front of the ego car, causing a glare. The SCENIC program describes three cars merging in a platoon-like manner where one car is following another car in front with some variations in distance between front and rear cars. The metric for distance perturbation is in meters. The images from Scenario 3 are shown in Figure 5.

Finally, Scenario 4 describes a set of scenes when the nearest car is abruptly switching into ego car's lane while another car on the opposite traffic direction lane is slightly intruding over the middle yellow line into the ego car's lane. Failure to detect these two cars out of four cars may potentially be safety-critical. The images from Scenario 4 are shown in Figure 6. The locations of all four cars, in Scenario 4 SCENIC program, are hard-coded with respect to ego car's location. The SCENIC program would have become much more interpretable had we described car locations with respect to lanes. The reason we had to code in this undesirable manner is due to the simulator as illustrated in Section 3.

2. Success and Failure Scenario Descriptions

The refined SCENIC programs characterizing success/failure scenarios are shown in Figure 7, 8, 9, and 10. The red/green parts of programs represent the rules automatically generated by our technique, which are cut and pasted to original SCENIC programs. These success/failure inducing rules are shown in Table 2 and 3 of the main paper. As mentioned in the Experiments section of the main paper, we generated new images using SCENIC programs that characterize failure scenarios. Examples of these images from failure scenarios are shown in Figure 11, 12, 13, and 14.

3. Limitation of SCENIC Description

The expressiveness and interpretability of SCENIC language is enhanced by leveraging detailed information about the map used in the simulator. For example, in Scenario 1 in Figure 1(a), we can directly refer to specific part of the road such as "curb" because we have already identified the curb regions in the map. And, description on placing of "otherCar" with respect to a spot on curb makes the program easily understandable to human. However, because GTA-V is not an open sourced simulator (in fact it is not even meant to be used as a simulator but is widely used for its realistic rendering), we could not parse out detailed map information such as regions of different lanes, location of traffic lights, etc. As the number of cars scaled, describing increasingly complicated geometric relations among cars without any reference points/objects/regions, such as lanes, became more challenging. As a result, we were limited to describe geometric relations in scenarios by only referencing cars, which resulted in much less understandable SCENIC programs, deprecating the use of SCENIC as a scenario description. We emphasize that this limitation is due to inaccessible map information of the simulator, not SCENIC.

```
param time = (6*60, 18*60)
ego = Car at -209.091 @ -686.231

spot = OrientedPoint on visible curb
badAngle = (-90,90) deg

otherCar = Car at spot,
                facing badAngle relative to ego.heading

require otherCar in ego.visibleRegion
require ((angle to otherCar) - ego.heading) < 0
require (distance from ego.position to otherCar) >= 5
require (distance from ego.position to otherCar) <= 20
```

(a) Scenario 1 SCENIC program

```
param time = (6*60, 18*60)
perturbation = (-10, 10)

ego = EgoCar at 123.757 @ -573.978

agent0 = Car visible,
                with roadDeviation resample(perturbation)

leftRight = Options([1.0, -1.0]) * (2, 3)
frontback = Options([1.0, -1.0]) * (1, 3)

agent1 = Car beyond agent0 by leftRight @ frontback,
     with roadDeviation resample(perturbation)

require (distance from ego.position to agent1) >= 5
require (distance from ego.position to agent0) <= 30
require (distance from ego.position to agent1) >= 5
require (distance from ego.position to agent1) <= 30
require agent0 in ego.visibleRegion
require agent1 in ego.visibleRegion
```

(b) Scenario 2 SCENIC program

Figure 1: Scenario 1 and Scenario 2

```
param time = (6*60, 18*60)

ego = Car at -576.5151 @ -62.7771
agent0 = Car offset by -2.5 @ 6.5,
           facing (40, 50) deg relative to roadDirection

distance_perturbation1 = (1.2, 1.5)
center1 = follow roadDirection from (front of agent0) for \
resample(distance_perturbation1)

agent1 = Car ahead of center1,
                facing (40, 60) deg relative to roadDirection

distance_perturbation2 = (1.2, 1.5)
center2 = follow roadDirection from (front of agent1) for \
resample(distance_perturbation2)

agent2 = Car ahead of center2,
                facing (-10, 10) deg relative to roadDirection

require agent0 in ego.visibleRegion
require agent1 in ego.visibleRegion
require agent2 in ego.visibleRegion
```

(a) Scenario 3 SCENIC program

```
param time = (6,18)*60
wiggle1 = (30 deg, 50 deg)
wiggle2 = (0 deg, 20 deg)

ego = EgoCar at -398.27272727273026 @ -222.81213535589416
agent0 = Car offset by (1.5,2) @ (8,9),
                with roadDeviation resample(wiggle1)

agent1 = Car offset by -1 @ (30,40)

agent2 = Car offset by (-8,-7) @ (14,15)

agent3 = Car offset by (-6,-5) @ (19,20),
                with roadDeviation resample(wiggle2)
```

(b) Scenario 4 SCENIC program

Figure 2: Scenario 3 and Scenario 4

Figure 3: Images generated using Scenario 1

Figure 4: Images generated using Scenario 2

Figure 5: Images generated using Scenario 3

Figure 6: Images generated using Scenario 4

```
param time = (6*60, 18*60)
ego = Car at -209.091 @ -686.231

spot = OrientedPoint on visible curb
badAngle = (-90,90) deg

otherCar = Car at spot,
                facing badAngle relative to ego.heading

require otherCar in ego.visibleRegion
require ((angle to otherCar) - ego.heading) < 0
require (distance from ego.position to otherCar) >= 5
require (distance from ego.position to otherCar) <= 20
require (otherCar.position.x>= -198.1)
```

(a) Refined Success Scenario 1 SCENIC program

```
param time = (6*60, 18*60)
ego = Car at -209.091 @ -686.231

spot = OrientedPoint on visible curb
badAngle = (-90,90) deg

modelNames = ['PRANGER']
models = []
for name in modelNames:
     models.append(CarModel.models[name])

otherCar = Car at spot,
                facing badAngle relative to ego.heading,
                with model Uniform(*models)

require otherCar in ego.visibleRegion
require ((angle to otherCar) - ego.heading) < 0
require (distance from ego.position to otherCar) >= 5
require (distance from ego.position to otherCar) <= 8.84
require (otherCar.position.x < -200.76)
```

(b) Refined Failure Scenario 1 SCENIC program

Figure 7: Refined Success and Failure SCENIC programs for Scenario 1, with red parts representing failure inducing rules and green parts representing the success inducing rules as shown in Table 2 and 3 of the main paper

```
param weather= Uniform('BLIZZARD','CLEAR','CLEARING', \
      'CLOUDS','EXTRASUNNY','FOGGY','OVERCAST', \
      'RAIN','SMOG','SNOW','SNOWLIGHT','THUNDER','XMAS')

param time = (7.5*60, 18*60)
wiggle = (-10, 10)

ego = EgoCar at 123.757 @ -573.978

modelNames = ['ASEA','BISON','BLISTA','BUFFALO',\
'DOMINATOR','JACKAL','NINEF','ORACLE']

models = []
for name in modelNames:
      models.append(CarModel.models[name])

agent0 = Car visible,
                with roadDeviation resample(wiggle),
                with model Uniform(*models)

leftRight = Options([1.0, -1.0]) * (2, 3)
frontback = Options([1.0, -1.0]) * (1, 3)

agent1 = Car beyond agent0 by leftRight @ frontback,
                with roadDeviation resample(wiggle)

require (distance from ego.position to agent0) >= 5
require (distance from ego.position to agent0) >= 11.3
require (distance from ego.position to agent1) >= 5
require (distance from ego.position to agent1) <= 30
require agent0 in ego.visibleRegion
require agent1 in ego.visibleRegion
```

(a) Refined Success Scenario 2 SCENIC program

```
param weather= Uniform('BLIZZARD','CLEAR','CLEARING', \
      'CLOUDS','EXTRASUNNY','FOGGY','OVERCAST', \
      'RAIN','SMOG','SNOW','SNOWLIGHT','THUNDER','XMAS')

param time = (7.5*60, 18*60)
wiggle = (-10, 10)

ego = EgoCar at 123.757 @ -573.978

agent0 = Car visible,
                with roadDeviation resample(wiggle)

leftRight = Options([1.0, -1.0]) * (2, 3)
frontback = Options([1.0, -1.0]) * (1, 3)

agent1 = Car beyond agent0 by leftRight @ frontback,
                with roadDeviation resample(wiggle)

require (distance from ego.position to agent0) >= 5
require (distance from ego.position to agent0) < 11.3
require (distance from ego.position to agent1) >= 5
require (distance from ego.position to agent1) <= 30
require agent0 in ego.visibleRegion
require agent1 in ego.visibleRegion
```

(b) Refined Failure Scenario 2 SCENIC program

Figure 8: Refined Success and Failure SCENIC programs for Scenario 2, with red parts representing failure inducing rules and green parts representing the success inducing rules as shown in Table 2 and 3 of the main paper

```
param time = (6*60, 18*60)

ego = Car at -576.5151 @ -62.7771
agent0 = Car offset by -2.5 @ 6.5,
          facing (40, 50) deg relative to roadDirection,
          with color CarColor.withBytes([(74.5,255),(0,255),\
(0,255)])

distance_perturbation1 = (1.2, 1.5)
center1 = follow roadDirection from (front of agent0) for \
resample(distance_perturbation1)

agent1 = Car ahead of center1,
               facing (40, 60) deg relative to roadDirection

distance_perturbation2 = (1.2, 1.5)
center2 = follow roadDirection from (front of agent1) for \
resample(distance_perturbation2)

agent2 = Car ahead of center2,
               facing (-10, 10) deg relative to roadDirection

require agent0 in ego.visibleRegion
require agent1 in ego.visibleRegion
require agent2 in ego.visibleRegion
require agent0.heading >= 220.3 deg
```

(a) Refined Success Scenario 3 SCENIC program

```
param weather = 'NEUTRAL'
param time = (6*60, 8*60)
ego = Car at -576.5151 @ -62.7771

agent0 = Car offset by -2.5 @ 6.5,
               facing (40, 50) deg relative to roadDirection

distance_perturbation1 = (1.2, 1.5)
center1 = follow roadDirection from (front of agent0) for
resample(distance_perturbation1)
agent1 = Car ahead of center1,
               facing (40, 60) deg relative to roadDirection

distance_perturbation2 = (1.2, 1.5)
center2 = follow roadDirection from (front of agent1) for
resample(distance_perturbation2)
agent2 = Car ahead of center2,
               facing (-10, 10) deg relative to roadDirection,
               with color CarColor.withBytes([(0,95),(0,255),\
(0,255)])

require agent0 in ego.visibleRegion
require agent1 in ego.visibleRegion
require agent2 in ego.visibleRegion
require agent0.heading <= 218 deg
```

(b) Refined Failure Scenario 3 SCENIC program

Figure 9: Refined Success and Failure SCENIC programs for Scenario 3, with red parts representing failure inducing rules and green parts representing the success inducing rules as shown in Table 2 and 3 of the main paper

```
param time = (6,18)*60
wiggle1 = (30 deg, 50 deg)
wiggle2 = (0 deg, 20 deg)

ego = EgoCar at -398.27272727273026 @ -222.81213535589416

modelNames0 =
['ASEA','BALLER','BLISTA','BUFFALO','DOMINATOR','JACKAL',\
'NINEF','ORACLE']
models0 = []
for name in modelNames0:
     models0.append(CarModel.models[name])

agent0 = Car offset by (1.5,2) @ (8,9),
                with roadDeviation resample(wiggle1),
                with model Uniform(*models0)

agent1 = Car offset by -1 @ (30,40)

agent2 = Car offset by (-8,-7) @ (14,15)

agent3 = Car offset by (-6,-5) @ (19,20),
                with roadDeviation resample(wiggle2)
```

(a) Refined Success Scenario 4 SCENIC program

```
param time = (6,18)*60
wiggle1 = (30 deg, 50 deg)
wiggle2 = (0 deg, 20 deg)

ego = EgoCar at -398.27272727273026 @ -222.81213535589416

modelNames0 = ['PATRIOT']
models0 = []
for name in modelNames0:
     models0.append(CarModel.models[name])

agent0 = Car offset by (1.5,2) @ (8,9),
                with color CarColor.withBytes([(0,255), \
                (92.25, 158.00), (0,84.25)]),
                with roadDeviation resample(wiggle1),
                with model Uniform(*models0)

modelNames1 = ['NINEF']
models1 = []
for name in modelNames1:
     models1.append(CarModel.models[name])

agent1 = Car offset by -1 @ (30,40),
                with model Uniform(*models1)

modelNames2 = ['BALLER']
models2 = []
for name in modelNames2:
     models2.append(CarModel.models[name])

agent2 = Car offset by (-8,-7) @ (14,15),
                with color CarColor.withBytes([(178,224), \
                (0, 255), (0,255)]),
                with model Uniform(*models2)


agent3 = Car offset by (-6,-5) @ (19,20),
                with roadDeviation resample(wiggle2)
```

(b) Refined Failure Scenario 4 SCENIC program

Figure 10: Refined Success and Failure SCENIC programs for Scenario 4, with red parts representing failure inducing rules and green parts representing the success inducing rules as shown in Table 2 and 3 of the main paper

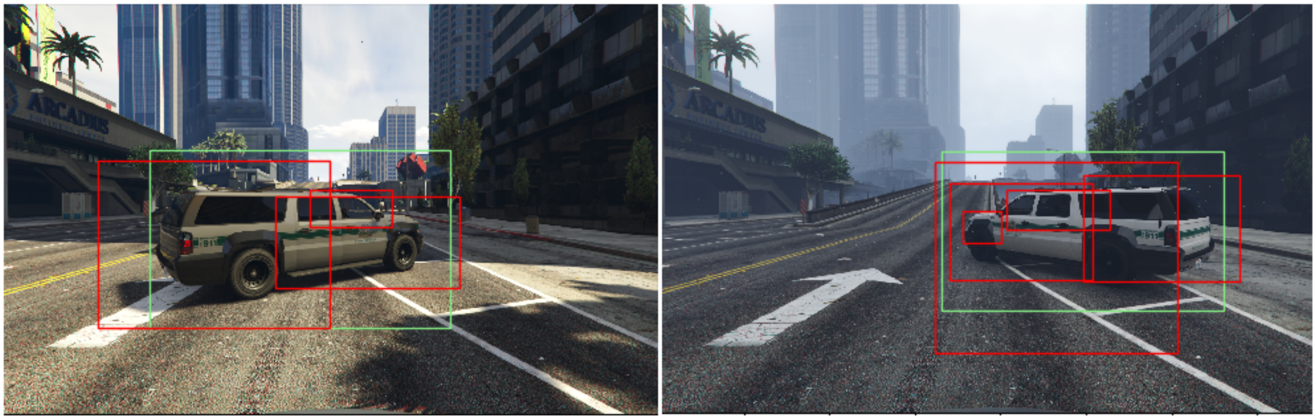

Figure 11: Failure images generated with Failure SCENIC Scenario 1 as shown in Figure 7 (b), with ground truth bounding boxes marked in green and prediction bounding boxes in red

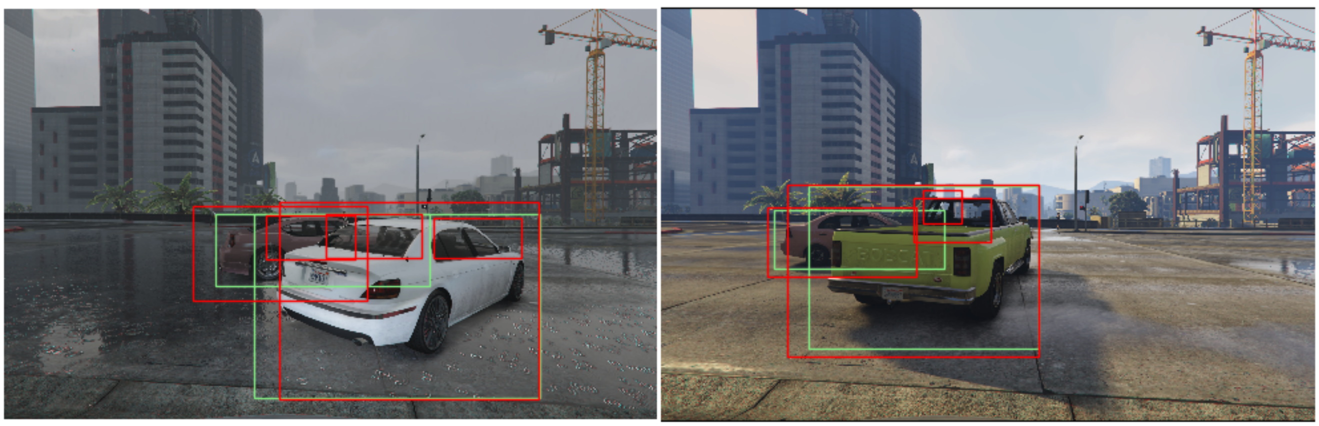

Figure 12: Failure images generated with Failure SCENIC Scenario 2 as shown in Figure 8 (b), with ground truth bounding boxes marked in green and prediction bounding boxes in red

Figure 13: Failure images generated with Failure SCENIC Scenario 3 as shown in Figure 9 (b), with ground truth bounding boxes marked in green and prediction bounding boxes in red

Figure 14: Failure images generated with Failure SCENIC Scenario 4 as shown in Figure 10 (b), with ground truth bounding boxes marked in green and prediction bounding boxes in red