A APPENDIX

A.1 BOUNCE SIMULATOR DESIGN DETAILS

We use PyMunk as physics engine and PyGame which provides backend for rendering. Creating a simulator is only necessary because we do not have full access to the original game simulator used by Code.org (which was written in JavaScript). We do not need to explicitly design simulator for other assignments if the original simulator is accessible. The simulator will be made available along with the dataset.

A.2 GENERALIZING OVER DIFFERENT THEMES

In Table 2, we evaluate each training strategy under different thematic combinations of visual elements of our game. Note that the "Mixed Theme" training strategy trained exactly on these 8 combinations. The table shows that vision-based data augmentations do not allow generalization across themes in our game.

	Baseline	Color Jitter	Cutout	Cutout Color	Gray Scale	Mixed Theme
Hardcourt	56.0 ± 20.8	59.0 ± 19.0	73.0 ± 21.9	77.0 ± 12.8	88.0 ± 12.5	62.0 ± 20.0
H-H-R	-44.0 ± 11.0	-46.0 ± 4.9	-40.0 ± 6.1	-36.0 ± 7.8	-42.0 ± 8.1	87.0 ± 6.7
H-R-H	-38.0 ± 9.8	-38.0 ± 8.1	-42.0 ± 6.0	-30.0 ± 14.5	-40.0 ± 6.1	71.0 ± 13.8
H-R-R	-32.0 ± 6.6	-46.0 ± 4.9	-42.0 ± 6.0	-42.0 ± 6.0	-42.0 ± 8.1	74.0 ± 15.5
R-H-H	-44.0 ± 7.8	-48.0 ± 3.7	-40.0 ± 11.3	-30.0 ± 16.2	-44.0 ± 5.6	80.0 ± 12.5
R-H-R	-44.0 ± 5.6	-44.0 ± 5.6	-42.0 ± 8.1	-44.0 ± 5.6	-46.0 ± 4.9	76.0 ± 15.7
R-R-H	-50.0 ± 0.0	-46.0 ± 4.9	-30.0 ± 9.5	-44.0 ± 7.8	-44.0 ± 5.6	65.0 ± 21.9
Retro	-46.0 ± 4.9	-50.0 ± 0.0	-32.0 ± 8.6	-42.0 ± 11.2	-42.0 ± 9.8	65.0 ± 18.8

Table 2: We show the average total reward for 10 runs from agent trained with different strategies. The confidence interval is computed with 90% confidence level. We evaluate the agent with eight theme variations. We refer to other settings in the order of background, paddle, ball (i.e., H-H-R refers to Hardcourt background, Hardcourt paddle, Retro ball).

A.3 GENERALIZING OVER DIFFERENT SPEEDS

Other than changing themes of game objects, Bounce also allows the change of game object's speed. There are five speed settings for each object. We trained all our agents under the "normal paddle" speed and "normal ball" speed setting. Since we have established that mixed-theme trained agent is best performing for all thematic differences, here we evaluate its ability to generalize to other speed settings.

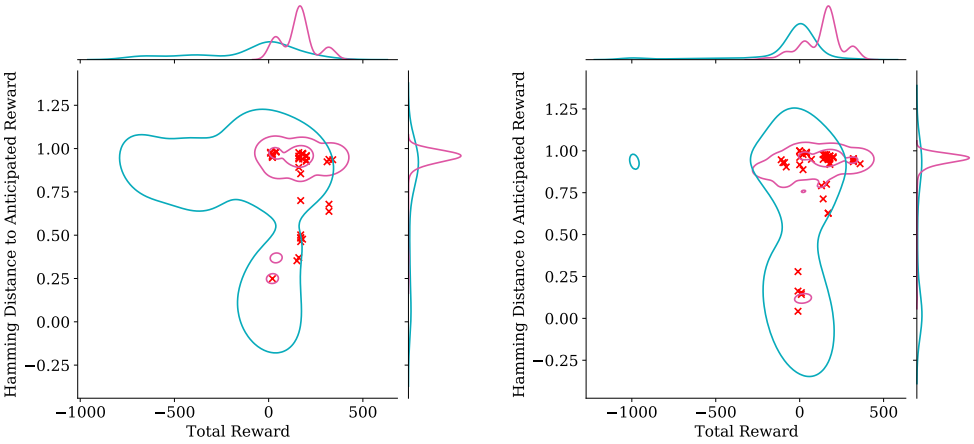
From Table 3, we note that unlike human players, who find more challenging to play when both ball and paddle moves "very fast", the agent can generalize easily to environments when both ball and paddle move faster than the speed it was trained on. We also didn't observe symmetry where the game play should be easy when both ball and paddle are at the same speed (i.e., "very slow paddle + very slow ball" receives very low reward, same for "slow paddle + slow ball").

A.4 DISTRIBUTIONAL SHIFT

In Figure 6, we visualize how our programs distribute in the feature space. Incorrect programs have a larger spread compared to correct programs in terms of both features. We can actually see that our

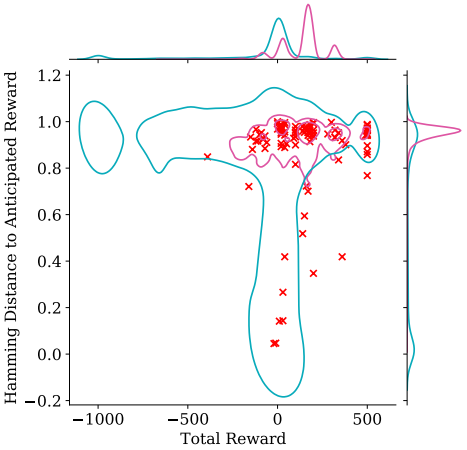
	very slow ball	slow ball	normal ball	fast ball	very fast ball
very slow paddle	7.0 ± 22.5	-24.0 ± 14.5	-17.0 ± 13.4	-14.0 ± 12.0	-4.0 ± 19.0
slow paddle	22.0 ± 24.9	51.0 ± 25.7	27.0 ± 22.7	40.0 ± 21.9	27.0 ± 31.8
normal paddle	64.0 ± 17.8	77.0 ± 10.2	82.0 ± 7.1	73.0 ± 18.1	75.0 ± 18.4
fast paddle	82.0 ± 9.4	93.0 ± 3.9	76.0 ± 15.0	87.0 ± 9.1	54.0 ± 24.8
very fast paddle	69.0 ± 26.4	78.0 ± 22.7	88.0 ± 6.6	94.0 ± 4.9	91.0 ± 5.8

Table 3: We show the 10-run average reward for the mixed-theme trained agent under different speed variations. We evaluate for maximal 2000 steps.

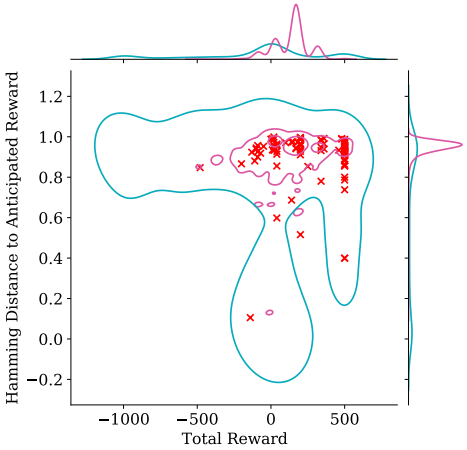


(a) Reference programs

(b) Head programs



(c) Body programs



(d) Tail programs

Figure 6: We show the kernel density plot when we visualize each program by their two features. Blue represents incorrect programs. Purple indicates correct programs. We overlay the error made by our best performing model as red x.

hand-crafted reference programs actually serve as a decent representation of the program space. We can also see that even with two features, we still can't completely tear apart correct and incorrect programs.

A.5 GOLD LABEL ANNOTATION SCHEMA

```
def main():
  sources = pickle.load(open(root + 'x.pickle', 'rb'))
  nDone = 0
  labelMap = {}

  for sourceId in tqdm(sources):
    rawStr = sources[sourceId]
    source = canonicalize(rawStr)
    labels = {}
    labelWhenRun(labels, source)
    labelWhenLeft(labels, source)
    labelWhenRight(labels, source)
    labelWhenPaddleCollided(labels, source)
    labelWhenWallCollided(labels, source)
    labelWhenBallInGoal(labels, source)
    labelWhenBallMissesPaddle(labels, source)
    labelMap[sourceId] = labels
    nDone += 1

    if nDone == N_TARGET: break
  pickle.dump(labelMap, open(root + 'labels.pickle', 'wb'))

def labelWhenBallMissesPaddle(labels, source):
  if not 'whenBallMissesPaddle' in source:
    addLabel(labels, 'whenMiss-noOpponentScore')
    addLabel(labels, 'whenMiss-noBallLaunch')
    return

  commands = source['whenBallMissesPaddle']

  if not 'incrementOpponentScore' in commands:
    addLabel(labels, 'whenMiss-noOpponentScore')
  if not 'launchBall' in commands:
    addLabel(labels, 'whenMiss-noBallLaunch')

  illegalCmds = [
    'incrementPlayerScore',
    'bounceBall',
    'moveLeft',
    'moveRight'
  ]

  nBalls = 0
  for cmd in commands:
    if cmd == 'launchBall':
      nBalls += 1
  if nBalls > 1:
    addLabel(labels, 'whenMiss-multipleBalls')

def labelWhenBallInGoal(labels, source):
  if not 'whenBallInGoal' in source:
    addLabel(labels, 'whenGoal-noPlayerScore')
    addLabel(labels, 'whenGoal-noBallLaunch')
    return

  commands = source['whenBallInGoal']
  if not 'incrementPlayerScore' in commands:
```

```
    addLabel(labels, 'whenGoal-noPlayerScore')

  if not 'launchBall' in commands:
    addLabel(labels, 'whenGoal-noBallLaunch')

  illegalCmds = [
    'incrementOpponentScore',
    'bounceBall',
    'moveLeft',
    'moveRight'
  ]
  for illegal in illegalCmds:
    if illegal in commands:
      addLabel(labels, 'whenGoal-illegalCmd')

  # do you have more than one ball?
  nBalls = 0
  for cmd in commands:
    if cmd == 'launchBall':
      nBalls += 1
  if nBalls > 1:
    addLabel(labels, 'whenGoal-multipleBalls')



def labelWhenWallCollided(labels, source):
  if not 'whenWallCollided' in source:
    addLabel(labels, 'whenWall-noBounce')
    return

  commands = source['whenWallCollided']

  if not 'bounceBall' in commands:
    addLabel(labels, 'whenWall-noBounce')

  illegalCmds = [
    'incrementOpponentScore',
    'incrementPlayerScore',
    'launchBall',
    'moveLeft',
    'moveRight'
  ]
  for illegal in illegalCmds:
    if illegal in commands:
      addLabel(labels, 'whenWall-illegalCmd')

def labelWhenPaddleCollided(labels, source):
  if not 'whenPaddleCollided' in source:
    addLabel(labels, 'whenPaddle-noBounce')
    return

  commands = source['whenPaddleCollided']

  if not 'bounceBall' in commands:
    addLabel(labels, 'whenPaddle-noBounce')

  illegalCmds = [
    'incrementOpponentScore',
    'incrementPlayerScore',
```

```
    'launchBall',
    'moveLeft',
    'moveRight'
  ]

  for illegal in illegalCmds:
    if illegal in commands:
      addLabel(labels, 'whenPaddle-illegalCmd')

def labelWhenRight(labels, source):
  if not 'whenRight' in source:
    addLabel(labels, 'whenRight-noMove')
    return

  whenDirCommands = source['whenRight']

  rightDelta = getRightDelta(whenDirCommands)

  if rightDelta == 0:
    addLabel(labels, 'whenRight-noMove')
  if rightDelta <= 0:
    addLabel(labels, 'whenRight-wrongMove')

  illegalCmds = [
    'bounceBall',
    'incrementOpponentScore',
    'incrementPlayerScore',
    'launchBall'
  ]
  for illegal in illegalCmds:
    if illegal in whenDirCommands:
      addLabel(labels, 'whenRight-illegalCmd')

def labelWhenLeft(labels, source):
  if not 'whenLeft' in source:
    addLabel(labels, 'whenLeft-noMove')
    return

  whenDirCommands = source['whenLeft']

  leftDelta = -1 * getRightDelta(whenDirCommands)
  if leftDelta == 0:
    addLabel(labels, 'whenLeft-noMove')
  if leftDelta <= 0:
    addLabel(labels, 'whenLeft-wrongMove')

  illegalCmds = [
    'bounceBall',
    'incrementOpponentScore',
    'incrementPlayerScore',
    'launchBall'
  ]
  for illegal in illegalCmds:
    if illegal in whenDirCommands:
      addLabel(labels, 'whenLeft-illegalCmd')


def labelWhenRun(labels, source):
  if not 'whenRun' in source:
```

```
    addLabel(labels, 'whenRun-noBallLaunch')
    return

  whenRunCmds = source['whenRun']
  if not 'launchBall' in whenRunCmds:
    addLabel(labels, 'whenRun-noBallLaunch')

  illegalCmds = [
    'bounceBall',
    'incrementOpponentScore',
    'incrementPlayerScore',
    'moveLeft',
    'moveRight'
  ]
  for illegal in illegalCmds:
    if illegal in whenRunCmds:
      addLabel(labels, 'whenRun-illegalCmd')

  nBalls = 0
  for cmd in whenRunCmds:
    if cmd == 'launchBall':
      nBalls += 1
  if nBalls > 1:
    addLabel(labels, 'whenRun-multipleBalls')

def getRightDelta(commands):
  rightDelta = 0
  for cmd in commands:
    if cmd == 'moveLeft':
      rightDelta -= 1
    if cmd == 'moveRight':
      rightDelta += 1
  return rightDelta

def addLabel(labels, key):
  labels[key] = True
  allLabels.add(key)
```