Class Project: Final Results

Nathaniel Reeves - CS 4300 Artificial Intelligence

Assignment: Project Report

Write a report that contains the PEAS assessment, a discussion of the search algorithm used, the variety of agent implementations, the variety of problem instances, the search results obtained, and conclusions about what can be learned from your project's results.

Note that this report is in effect your final exam.

PEAS Assessment

Spider Solitaire is a card game that requires strategic planning and decision-making to organize two shuffled decks of four suits of thirteen cards (Ace to King) each into complete sequences. The primary objective is building eight foundation piles, one for each suit, from King to Ace in descending order (King to Ace).

A useful AI agent would analyze the current layout of cards, determine the best moves to make, and optimize the game-play strategy to complete the game efficiently. The optimal result would be to maximize the number of completed sequences while minimizing the number of moves and time taken, all while adhering to the game's rules and constraints regarding card movement.

The Performance Measure:

The performance measure for the Spider Solitaire agent will be calculated based on the following criteria, with the sum of these factors yielding a positive score for a good agent. The best agent will have the highest performance measure. The components of the performance measure are as follows:

- **10 * (number of face-down cards turned face up):** For each initial face-down card that gets turned over, the agent is awarded 10 points.
- **15 * (number of piles that contain zero face-down cards:**
- **2 * (number of cards placed atop the next higher card of the same suit)**
- **50 * (number of completed suits)**
- **2 * (number of completed suits after the first three):** If the game ends with 4 or more completed suits still in the tableau, add 2 points for each suit after the first three.

(Scoring system from Sun Microsystems implementation of Spider Solitaire in 1989) The highest possible score using the Sun Microsystems implementation is 1000 points.

The following penalties are added to the performance measure to focus on more efficient agents.

- **-1 * (number of moves taken)**

I will also consider an additional performance measure multiplier that rewards more points based on the number of unused bank piles. This might incentivize the agents to make drawing from the bank a less desirable action without giving a penalty to the agent.

A move is defined as:

- Placing a card or group of cards
- Drawing from the bank

The Environment is:

- **Observability: Partially observable.** The agent can see the cards currently in play and how many cards are in each pile. It can see the number of banks it has left and the number of suits it has been able to complete. It cannot see the value and suits of cards that are face down in both the bank and the piles.
- **Uncertainty: Semi-Deterministic.** Once the agent makes a move with turned-over cards, the outcome is predictable, but when the agent turns over a card by revealing a card in a pile or pulling from the bank only a probability can be calculated for what that card might be.
- **Duration: Sequential.** Each move affects the state of the game, however, the agent only needs to reassess the best strategy when a new card is revealed.
- **Stability: Static.** The layout of cards remains unchanged until the agent acts, but the agent must adapt its strategy based on the current state of the game and the sequence of available moves.
- **Granularity: Discrete.** The game operates in discrete actions where each move, card draw, and card placement are counted as a separate event. The agent will consider each action distinctly.
- **Participants: Single-agent.** The agent plays independently without other participants. Its decisions are based solely on the current game state and its internal strategies.
- **Knowledge: Known.** The rules of Spider Solitaire are well-defined and the mechanics of card movement, win conditions, and available strategies are understood. The agent will leverage this knowledge to inform its decision-making process, and further details regarding optimal strategies will be documented.

The Actions (Actuators):

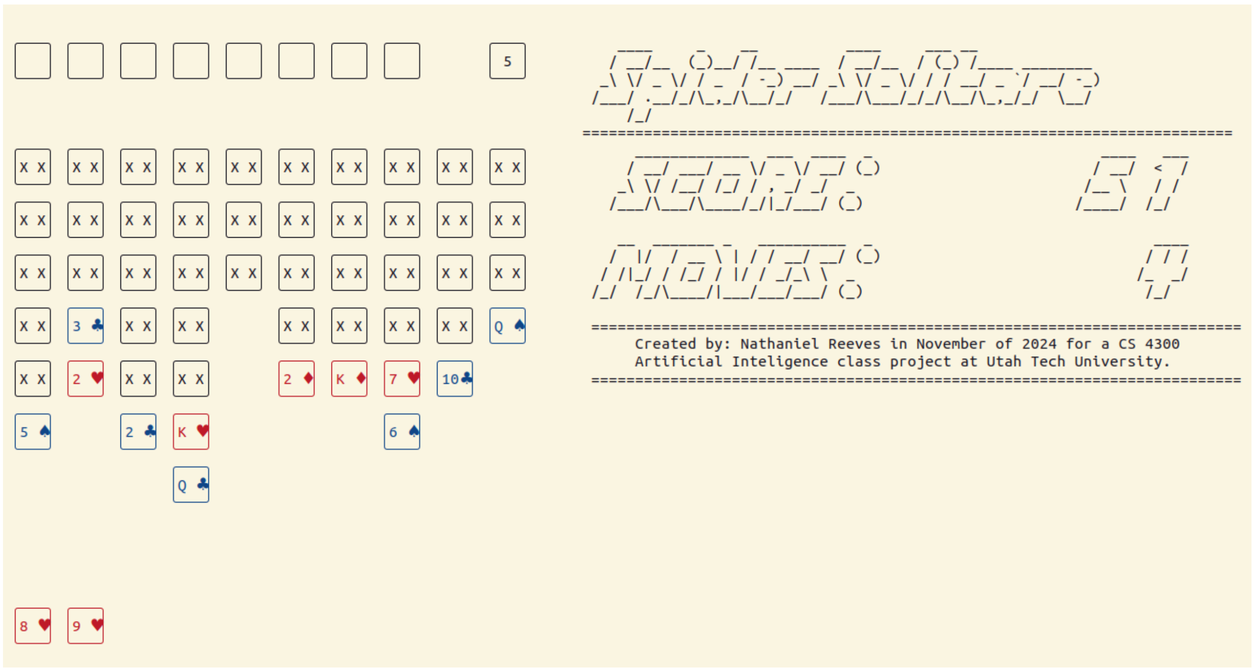
- **Pick Up Card (or card group):** The agent can pick up a card or an entire group of cards from a card pile if the cards are in the same suit and are in descending order. Only one card or card group can be held by the agent at a time. If the agent picks up a card group, the group must remain in the same order it was picked up.
- **Place Card (or card group):** The agent can place a card or card group onto a card pile if the bottom-most card in the group is one value less than the topmost card on the selected card pile. If the agent is placing a card group, the entire group must be placed at once in the same order it was picked up.

- **Play Bank Cards:** The agent can deal one card on top of each pile (including empty piles) from the bank. All banks must get one new card in this action due to the rules of Spider Solitaire.

The Percepts (Sensors):

- The current layout of the card piles:
    - This includes the values and suits of all visible cards in each pile and,
    - The turned over (hidden) cards.
- Order and number of cards in the agent's hand.
- Number of completed runs.
- Number of cards left in the bank.
- The move count of the agent.
- The game score (see the performance measure).

Spider Solitaire Environment



```
Created by: Nathaniel Reeves in November of 2024 for a CS 4300
Artificial Inteligence class project at Utah Tech University.
```

Video: Agent v4 Completeting One Suit Seed 180 Fast.webm

Action Space	`Discrete(number_of_piles * 2 + 1) (default 10 piles)`
Observation Space	`card_space = spaces.MultiDiscrete( [number_of_ranks + 1, number_of_suits + 1, 2] )` `pile_space = spaces.Sequence(card_space, stack=True)` `completed_suit_space = spaces.Sequence(card_space, stack=True)` `Dict({ "bank": spaces.Sequence(card_space, stack=True) "completed_suits": spaces.Dict({ f"completed_suit_{i+1}": completed_suit_space for i in range(num_completed_suits) }) "hand": spaces.Sequence(card_space) "piles": spaces.Dict({ f"pile_{i+1}": pile_space for i in range(self._num_piles) }) "pickup_pile_index": spaces.Discrete(self._num_piles + 1), "num_moves": spaces.Discrete(600), "score": spaces.Discrete(2000, start=-999) })`
Import	`gymnasium.make("spider_solitare/SpiderSolitare-v0")`

Description

The game starts with a standard spider solitaire setup. Two decks of 52 cards (13 ranks per suit) were shuffled, then divided into 10 piles with 50 cards dealt to the bank. The top card of each pile is turned face up while the remaining cards are face down.

The goal of the game is to sort out all the cards by their suit and rank. A winning game has each pile cleared of cards and 8 completed suits of 13 cards.

Observation Space

Each card is represented by a three tuple.

- Card[0] represents the rank (from 1-13)
- Card[1] represents the suit (from 0-4)
- Card[2] represents a faceup/facedown state (0 for face down, 1 for face up)

A hidden card will have a 0 in each position.

The observation space is a dictionary containing the following sub-spaces:

- **Bank** - contains all cards that are not currently in the game.
- **Completed Suits** - contains all cards that are found to be in a completed suit. A default game has 8 completed suit slots with a capacity of 13 cards each.
- **Hand** - This is a stack data structure of cards. Cards that are picked up are pushed to this stack. The hand space dumps all cards during a putdown action.
- **Piles** - A default game has 10 piles. Each pile is a stack data structure containing each card. Cards can be popped onto a pile from the hand or they can be popped off the pile to the hand. Each pile is listed by the "pile_{index}" key where the index represents piles from 1 to 10. There is no pile 0.
- **Pickup Pile Index -** The index of the most recent pickup action. Since cards can only be picked up from one pile at a time, any pickup action will update this flag with the pile index selected. Put down actions update the Pickup Pile Index back to zero indicating to the agent that a card can again be picked up from any pile.
- **Num Moves -** The total number of moves taken by the agent. A move is defined by either a put-down action or a draw-bank action.
- **Score -** the current game score (Sun Microsystems 1989) minus the number of moves taken.

Action Space

The normal rules of spider solitaire apply. You can only move cards that are face up from pile to pile. Cards can only be placed on each other if they are one rank less than the card below. If a face-up card is moved from a pile with face-down cards, the top face-down card is turned over. The bank can deal one card on top of each pile, (this game allows for bank deals with empty piles)

The environment accepts actions as numbers between {0, 2 times the number of piles}.

- Action 0 will always be the draw from bank action. Draw bank actions are only allowed if there are cards in the bank and if there are zero cards in the hand space.
- Actions 1 through the number of piles are pickup actions. Any time a card is picked up, it is placed in the hand space in the order it was picked up. The pickup_pile_index is updated to the index of the pile most recently picked up from. The environment only allows additional pickup actions that are from the same pickup_pile_index, (aka: you can only pick up cards from one pile at a time).
- Actions 1 + number of piles through 2 * the number of piles is designated as a put down action, where the action is the index (minus the number of piles) the cards in the hand will be put down. Put-down actions are only valid if there are cards in the hand space. A put-down action will clear all cards from the hand and update the pickup_pile_index to zero.

Starting State

The game will begin with the cards already shuffled and dealt into the piles and bank.

- These piles are listed as “pile_1” through “pile_10”
    - (with 10 being the default number of piles).
    - Call cards in each pile will be hidden except the top card in each pile.
- The bank will have 50 cards in a standard game.
    - All 50 cards are also hidden.
- The remaining spaces, (i.e. the hand and completed_suits) will have zero cards.
- The num_moves, score, and pickup_pile_index are initialized to zero.

Reward

Each time a card or group of cards are moved from one pile to another, or if a bank deal takes place, that counts as one move. Scores are calculated after each move following the scoring system from Sun Microsystems' implementation of Spider Solitaire in 1989 with the addition of move count penalties(see The performance Measure in the PEAS assessment). The highest possible score using the Sun Microsystems implementation is 1000 points. The space allows for a score of -999 to 1000 due to the potential for move penalties.

The environment will check each pile after each move for completed suits. A completed suit is one card of each rank of the same suit in order of rank from highest to lowest. Completed suits are moved to the completed suit space as they are found and additional score points are added.

Episode End

The game/episode will terminate if the agent takes an invalid action.

The game/episode will truncate after 750 actions are taken by default.

Arguments

```
gym.make(
  'spider_solitare/SpiderSolitare-v0',
  render_mode,
  num_suits,
  max_episode_steps,
  theme
)
```

- render_mode (None, ‘ansi’)
    - None - no render
    - ‘ansi’ - render the game in the terminal with ascii
- num_suits (1, 2, 4) - Number of suits per deck.
- max_episode_steps
    - Number of actions allowed before truncating the episode.
    - Default 750 steps.
- theme (‘light’, ‘dark’)

    - changes the color scheme in the ansi render for dark terminals or light terminals.
    - default ‘dark’

References

Wikimedia Foundation. (2024, August 15). Spider (solitaire). Wikipedia.
https://en.wikipedia.org/wiki/Spider_(solitaire)

Version History

- v0: Initial version release

Agent & Search Algorithm

In total, I was able to make one agent with four iterations. As I made these iterations, it quickly became apparent that the strategies this agent used worked great for single-suit spider solitaire, however, this strategy was soon met with diminishing returns for higher suit counts. This is duly noted in the results section of this report. The following descriptions of this custom agent with the subsequent iterations.

Agent V1

The initial concept for this search algorithm was a variation of MiniMax search. Max and Min nodes were replaced with Pickup and Putdown nodes. The main goal of the algorithm was to ensure each sequence of actions ended with a valid Putdown action. While this would make the agent somewhat short sighted, it would ensure all actions would be valid.

```
def HANDLE_PICKUP_NODE(self, node):
    # Unpack Node
    action = node.action
    observation = node.observation
    depth = node.depth

    if depth == self.max_depth:
        return

    actions = self.model.ACTIONS(observation)
    for action in actions:
        if action == 0:
            continue
        if action > 0 and action <= self.num_piles:
            new_node = PickupNode()
            new_node.action = action
            new_node.observation = self.model.RESULT(observation, action)
            new_node.depth = depth + 1
            new_node.parent_node = node
            self.HANDLE_PICKUP_NODE(new_node)
        if action > self.num_piles:
            new_node = PutdownNode()
            new_node.action = action
            new_node.observation = self.model.RESULT(observation, action)
            new_node.parent_node = node
            new_node.score = self.model.EVALUATE(new_node.observation)
            new_node.goal = self.model.GOAL_TEST(new_node.observation)
            new_node.depth = depth + 1
            self.putdown_nodes.append(new_node)

def custom_search(self, observation):
    self.num_piles = self.model.NUM_PILES(observation)

    actions = self.model.ACTIONS(observation)
    self.max_score = self.model.EVALUATE(observation)

    # Generate Pickup Nodes
    process_nodes = []
    for action in actions:
        if action == 0:
            continue
        if action > 0 and action <= self.num_piles:
            pickup_node = PickupNode()
            pickup_node.action = action
            pickup_node.observation = self.model.RESULT(
                           observation, action)
            pickup_node.depth = 1
            process_nodes.append(pickup_node)
        if action > self.num_piles:
            continue

    # Process Pickup Nodes
    for node in process_nodes:
        self.HANDLE_PICKUP_NODE(node)

    # Filter all putdown nodes that have a score less than the max score
    filtered_putdown_nodes = [node for node in self.putdown_nodes if
node.score >= self.max_score]

    if len(filtered_putdown_nodes) == 0:
        self.actions = [0]
        return

    # Select the best putdown node and generate the action sequence
    selected_node = self.select_best_putdown_node(filtered_putdown_nodes)
    computed_actions = self.generate_action_sequence(selected_node)

    # Cycle through actions until a new valid sequence is found
    while True:
        filtered_putdown_nodes.remove(selected_node)
        if len(filtered_putdown_nodes) == 0:
            computed_actions = [0]
            break
        if self.validate_actions(observation, computed_actions):
            break
        selected_node =
self.select_best_putdown_node(filtered_putdown_nodes)
        computed_actions = self.generate_action_sequence(selected_node)

    self.actions = computed_actions
```

Agent V2

In this version, I aimed to tackle the two major faults in the first version. Namely, the getting bad searches issue and the bank dumping issue.

- Bad Search
    - After some digging, I soon realized that the goal test function built into the environment did not give enough rewards for smaller moves. Small moves are necessary to help the game progress forward. Getting inspiration from the heuristic concept of A Star seach, I created a evaluation function that utalized the scoring system built into the environment.
- Bank Dump
    - Using the score evaluation, I was able to create a limit for bad searches. Initially, I set the limit really low, with only three bad searches allowed. A bad search is defined as a sequence of actions that cause the game score to become less than the initial score before the sequence of actions. As soon as the agent reached the bad search limit, it would call a draw bank action and begin searching again.

These tweaks to my algorithm didn't bring any immediate gains, however I new what future improvements could be made with the new infrastructure.

Agent V3

This agent calculates bonus points and extra penalties that are added to the evaluation function output. The fine grain rewards and penalties guide the search algorithm to better action sequences. Here is the bonus function in a nutshell:

- If a pile is empty, minus 10 points
- Count the number of streaks in each game pile:
    - (A streak is a small run of two or more cards of the same suit)
    - If there is a streak, add streak length * the sum of each card rank in the streak
- If all cards in a pile are face up, double the bonus points.

The bonus points are completely virtual. These bonus points incentivize the agent to flip over face-down cards and sort them into their suits by rank.

This change saw a massive improvement in the agent performance, however the bonus points had an unexpected side effect. The agent would easly solve the first 1-4 suits then suddenly halt on the final 3-4 suits.

Agent V4

The problems left from the previous version were a bit more tricky to solve. After scanning through several game seeds, I was able to select one of these almost-solved games to run version 4 against, tweaking the bonus points function until it managed to solve the game. This seed happened to be seed 180 in a single-suit game.

The issue stemmed from the artificial bonus points incentivising the agent to leave the game incomplete. It seems the bonus points help the agent get through half the game only for the bonus points to stop the agent from finishing the game.

There were three main changes that were made to cerb the effect of bonus points in the final game.

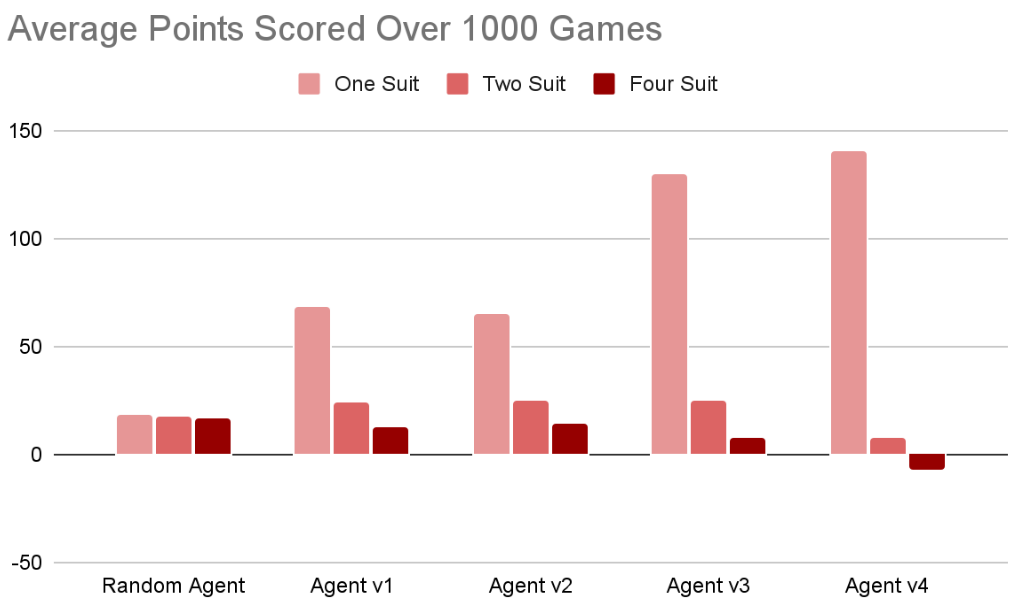
Dial back bonus points - Instead of awarding bonus points for each streak found in a pile, the new function would only awarded the longest streak of each pile. Bonus points would not be given for piles that contain a king. This was to incentivize the agent to remove kings from the game.

Reduce the bonus points awarded as the game progressed.

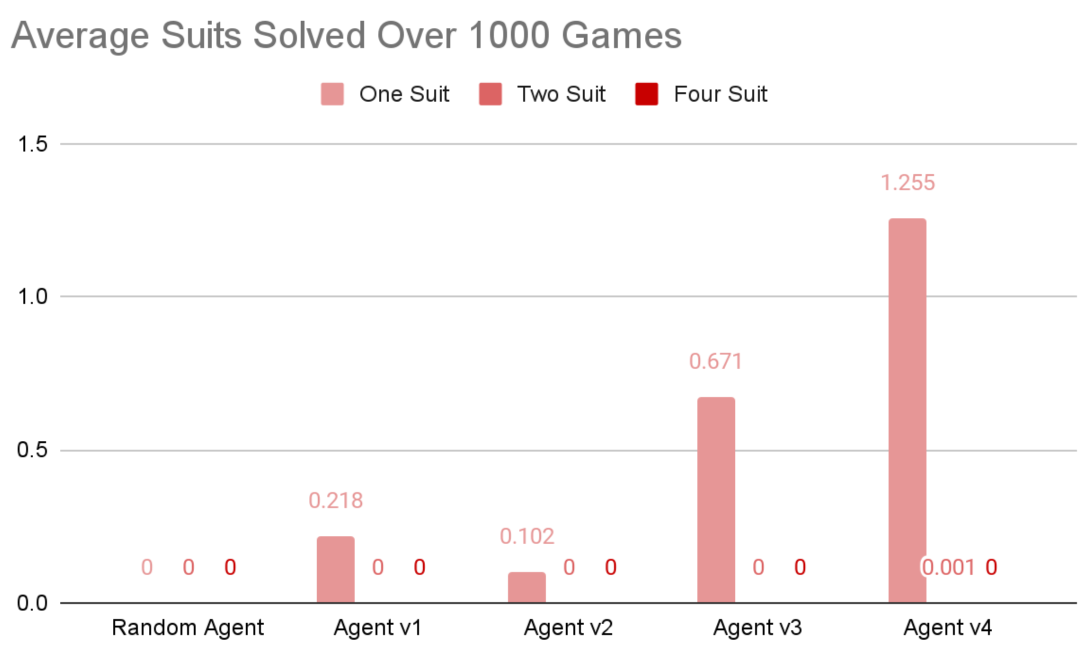
Increased the bad search limit as the game progressed. - This allowed the agent to experiment more as the game progressed until it was able to solve the seed 180 game.

After these changes, I was hopeful to see another big bump in average preformace. In the end, version four didnt not make as large of an increase from version three, however I was happy to finally have an agent that could finish a game of Spider Solitaire.

Final Results



Ave Scores from 1000 Games	**One Suit**	**Two Suit**	**Four Suit**
Random Agent	18.239	17.346	16.928
Agent v1	67.981	24.396	12.395
Agent v2	64.658	24.554	14.242
Agent v3	129.536	24.742	7.733
Agent v4	140.386	7.553	-6.813



Average Suits Solved Over 1000 Games	**One Suit**	**Two Suit**	**Four Suit**
Random Agent	0	0	0
Agent v1	0.218	0	0
Agent v2	0.102	0	0
Agent v3	0.671	0	0
Agent v4	1.255	0.001	0

Conclusion

During this project, and struggling with a good search algorithm, I discovered two interesting details:

- Spider Solitaire has been categorized in the NP-Complete category of Computer Science problems. These kind of problems are great applications for AI algorithms to tackle.
- The better my agent got at single-suit spider solitaire, the worse it got at two and four-suit solitaire. Its possible higher difficulty spider solitaire require a completely different strategy for an AI to solve.

Over all this was a fun and insightful project and a good final for this class. I feel that I have gained better experience in the field of Computer Artificial Intelligence.